Simplified Derandomization of BPP Using a Hitting Set Generator

Oded Goldreich, Salil Vadhan, and Avi Wigderson

Abstract. A hitting-set generator is a deterministic algorithm that generates a set of strings such that this set intersects every dense set that is recognizable by a small circuit. A polynomial time hitting-set generator readily implies $\mathcal{RP} = \mathcal{P}$, but it is not apparent what this implies for $\mathcal{BPP}$. Nevertheless, Andreev *et al.* (ICALP'96, and JACM 1998) showed that a polynomial-time hitting-set generator implies the seemingly stronger conclusion $\mathcal{BPP} = \mathcal{P}$. We simplify and improve their (and later) constructions.

Keywords: Derandomization, RP, BPP, one-sided error versus two-sided error.

This work is considered the final version of [7]. An early version of this work appeared as TR00-004 of *ECCC*. The current revision is quite minimal.

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 $\mathcal{BPP} = \mathcal{P}$?*

1.1 An Affirmative Partial Answer

The breakthrough paper of Andreev *et al.* [2] (and its sequel [3]) gave a natural setting in which the answer to the foregoing question 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 $\mathcal{RP}$ set). Having such a generator that 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 [2] was that such a generator (which immediately yields deromization of 1-sided error algorithms) actually suffices for derandomizing 2-sided error algorithms. In particular, the existence of polynomial-time hitting set generators implies $\mathcal{BPP} = \mathcal{P}$.

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

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.* [2] is the following:

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

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

Corollary 3 (Theorem 2, specialized [2]): *If* G *runs in polynomial time, then* $\mathcal{BPP} = \mathcal{P}$*.*

1.2 In Quest of Simplifications

Our main result is a simple proof of Theorem 2. Explaining what "simple" means is not so simple. We start by explaining how the given generator (assumed in the hypothesis of Theorem 2) is used in [2] (and in subsequent works [3,4]) to derandomize $\mathcal{BPP}$. Indeed, later proofs (of [3] and then [4]) were much simpler than [2], but while proving Corollary 3, they fell short of proving Theorem 2.[2]

Warning: The following discussion is carried out on an intuitive (and somewhat vague) level. Readers who do not like such discussions may skip the rest of the introduction, and proceed directly to the formal proof (presented in Sections 2 and 3).

The Two Different Uses of the Hitting Set Generator in [2]. The proof in [2] uses the generator in two ways. The first use is literally as a producer of a hitting set for all sufficiently dense and efficiently recognized sets. The second use (which is more subtle) is as a hard function. Indeed, observe that the existence of such

[1] In other settings, (pseudorandom) generators are defined as outputting a single string. In terms of Definition 1 this means that, on input an index $i \in \{1, ..., |\mathrm{G}(n, s)|\}$ (viewed as a seed), the generator outputs the i^{th} string in $\mathrm{G}(n, s)$. However, in the current context, the current convention (which in the standard terms means considsering the set of all possible outputs of the generator) seems simpler to work with.

[2] We note, however, that both [3] and [4] use their techniques to study other aspects of the relationship between one-sided and two-sided error (i.e., aspects not addressed by Theorem 2). In particular, Buhrman and Fortnow [4] resolve the promise-problem analogue of the question "Does $\mathcal{RP} = \mathcal{P}$ imply $\mathcal{BPP} = \mathcal{P}$?" in the positive. See Section 1.3.

a generator G immediately implies the existence of a function on $O(\log \mathrm{t_G}(s))$ bits that is computable in time $\mathrm{t_G}(s)$ but cannot be computed by circuits of size s (or else a contradiction is reached by considering a circuit that that accepts a vast majority of the strings that are not generated by G).[3] These two uses of G are combined in a rather involved way for the derandomization of $\mathcal{BPP}$.

It is interesting to note that for the case of $\mathrm{t_G}(s) = \mathrm{poly}(s)$, the aforementioned hard function can be plugged into the pseudorandom generator of [8], to yield $\mathcal{BPP} = \mathcal{P}$ as in Corollary 3. (Note, however, that [8] was not available to the authors of [2] at the time (the two papers are independent).) Moreover, [8] is far from "simple", it does use the computational consequence (which we are trying to avoid), and anyhow it is not strong enough to yield Theorem 2.

The Two-Level Use of the Hitting Set Generator in [3]. A considerably simpler proof was given in [3]. 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 $\mathrm{poly}(\mathrm{t_G}(\mathrm{t_G}(\mathrm{poly}(n))))$.

On the one hand, this is not too bad. For the interesting case of $\mathrm{t_G}(s) = \mathrm{poly}(s)$ (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 $\mathrm{t_G}$ grows moderately such that $\mathrm{t_G}(\mathrm{t_G}(n)) = 2^n$, then we have as assumption a highly nontrivial derandomization of $\mathcal{RP}$, but the consequence is a completely trivial derandomization of $\mathcal{BPP}$.

In our opinion, the best way to understand the origin of the iterated application of the function $\mathrm{t_G}$ in the aforementioned result is explained in the recent paper of [4], which further simplifies the proof of [3]. They remind the reader that the proofs [9,10] 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 $\mathcal{BPP}$ set, the "witnesses" for the existential quantifier are abundant. Put differently, $\mathcal{BPP} \subseteq \mathcal{RP}^{\mathrm{pr}\mathcal{RP}}$, where $\mathrm{pr}\mathcal{RP}$ 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 $\mathcal{RTIME}(\mathrm{t_G}(\mathrm{poly}(n))$ machine, which can again be derandomized (using hitting sets for $\mathrm{t_G}(\mathrm{poly}(n))$ 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 the $\mathcal{BPP} \subseteq \Sigma^2 \cap \Pi^2$ result. So another route has to be taken in order to prove Theorem 2 in a similar "direct" (or "black-box") manner, but without incurring the penalty arising from this two-level recursion.

Our Proof. We eliminate the two-level recursion, obtaining a single application of the hitting set generator, by "increasing the dimension to two" in the following

[3] For example, consider a circuit C that accepts a $(2\log_2 \mathrm{t_G}(s))$-bit string, z, if and only if z is not a prefix of any string in $\mathrm{G}(n,s)$.

sense. Inspired by Lautemann's proof [9] of $\mathcal{BPP} \subseteq \Sigma^2 \cap \Pi^2$, we consider, for each input to a given $\mathcal{BPP}$ algorithm that uses $\ell(n)$ random coins, a $2^{\ell(n)}$-by-$2^{\ell(n)}$ Boolean matrix such that the $(a,b)^{\mathrm{th}}$ entry represents the decision of the algorithm when using the random pad $a \oplus b$.[4] 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 that are monochromatic, and decide accordingly. The correctness and efficiency of the test are spelled out in Lemma 6, which is essentially captured by the following simple Ramsey-type result.

Proposition 4 (log-size dominating sets): *For every n-vertex graph, either the graph or its complement has a dominating set of size* $\lceil \log_2(n+1) \rceil$. *Furthermore, such a set can be found in polynomial time.*

(Proposition 4 is seemingly new and may be of independent interest.)[5]

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.

1.3 Perspective

As described above, Buhrman and Fortnow [4] 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 $\mathrm{pr}\mathcal{RP} = \mathrm{pr}\mathcal{P}$ implies $\mathrm{pr}\mathcal{BPP} = \mathrm{pr}\mathcal{P}$, 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 classes of decision problems, $\mathcal{RP}$ and $\mathcal{BPP}$. (In [4], it is shown that such an extension cannot relativize.) The second possible extension is to "scale" the result upwards. In fact, 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 proved in [2] and in 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

[4] In a preliminary version of this work [7], we considered a different matrix such that its $(a,b)^{\mathrm{th}}$ entry represents the decision of the algorithm when using the 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, where this strategy was inspired by [6]. The main simplification here is in avoiding this strategy.

[5] Proposition 4 corresponds to the special case of Lemma 6 that refers to symmetric matrices. Note that this special case actually suffices for our application.

inputs of length n; similarly, define $s = s(n)$ such 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 next, utilizes a hitting-set generator G as defined earlier (i.e., in Definition 1).

Derandomization procedure: On input $x \in \{0,1\}^n$, let 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 $\mathsf{N} \stackrel{\text{def}}{=} \mathsf{N}_{\mathrm{G}}(s)$ and each e_i is in $\{0,1\}^\ell$.
2. Construct an N-by-N matrix, $M = (v_{i,j})$, such 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 the procedure guaranteed by Lemma 6 (of Section 3 (below)), determine whether for every ℓ columns there exists a row on which all these columns have 1-value. If this 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 the foregoing description, by itself, does not establish the correctness of the procedure. Neither did we specify how to efficiently implement Step 3, To that end we use a general technical lemma (indeed Lemma 6) that implies that *it cannot be the case that both Eq. (1) and Eq. (2) hold.* Furthermore, this lemma asserts that one can efficiently determine which of the two conditions does not hold. These aspects are deferred to the next section. But first we prove the foregoing implications.

Proposition 5 (on the matrix constructed in Step 2): *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 next 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, it holds that

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

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

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

Let $\overline{y} = (y_1, ..., y_\ell)$, and consider the circuit $C_{x,\overline{y}}(z) \overset{\text{def}}{=} \wedge_{i=1}^{\ell}(C_x(y_i \oplus z) = \chi_L(x))$. Then, by the Eq. (3), it holds that $\mathrm{Pr}_z[C_{x,\overline{y}}(z) = \chi_L(x)] \geq 1/2$. 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 $H = \mathrm{G}(\ell\,,\ \ell \cdot s)$ must contain a string z such that $C_{x,\overline{y}}(z) = \chi_L(x)$.

The foregoing holds for any $\overline{y} = (y_1, ..., y_\ell)$. Thus, for every $y_1, ..., y_\ell \in H \subseteq \{0,1\}^\ell$, there exists $z \in H$ such that $A(x, y_i \oplus z) = C_x(y_i \oplus z) = \chi_L(x)$ holds for every $i \in [\ell]$. Thus, we have proved that, for every ℓ rows in M, there exists a column (in M) 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 $z_1, ..., z_\ell \in \{0,1\}^\ell$, it holds that

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

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

Digest. The foregoing procedure is a simplified version of the procedure given in the preliminary version of this work [7]. Specifically, inspired by [6], the argument in [7] relies on the explicit constructions of extractors for drastic error reduction of the $\mathcal{BPP}$ algorithm. Here, we only use a mild (and trivial) error reduction. This difference stems from the fact that the matrix considered in [7] is different (i.e., the $(a, b)^{\text{th}}$ entry in the matrix considered in [7] represents the decision of the algorithm when using the random pad $a \circ b$ (rather than $a \oplus b$)). In contrast, Steps 1 and 3 of the foregoing derandomization procedure are identical to the steps in [7], and so is Lemma 6. Thus, our argument relies on two essential ingredients: The first ingredient, adopted from [3], 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, which consists of using extractors as in [7], 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 before, it is not even clear that Eq. (1) and Eq. (2) cannot hold simultaneously. This is asserted next.

Lemma 6 (a generic technical lemma): *For every $k \geq \log_2(n+1)$, every n-by-n Boolean matrix either has k rows whose bit-wise disjunction is the all 1's row, or k columns whose bit-wise conjunction 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 $n = \mathsf{N} = \mathsf{N}_{\mathrm{G}}(s)$ and $k = \ell$, and furthermore $n < 2^{\ell}$ (since there is no point in having $\mathrm{G}(\ell, \cdot)$ contain more than $2^{\ell-1} + 1 < 2^{\ell}$ strings of length $\ell > 1$). The claim then follows by applying the following corollary to Lemma 6.

Corollary 7 (corollary to Lemma 6): *For every n-by-n Boolean matrix and every $k \geq \log_2(n+1)$, it is impossible that both the following conditions hold:*

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

Furthermore, assuming one of the foregoing conditions holds, deciding which one holds can be done in (deterministic) *polynomial-time.*

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

Proof of Lemma 6: Let $B = (b_{r,c})$ be an arbitray n-by-n Boolean matrix, and consider the following iterative procedure, initiated with $C_0 = [n]$ and $R = \emptyset$. For $i = 1, 2, ...$, take a row r not in R that has at least $|C_{i-1}|/2$ 1's in C_{i-1} (i.e., $r \in [n] \setminus R$ such that $|\{c \in C_{i-1} : b_{r,c} = 1\}| \geq |C_{i-1}|/2$). Add r to R, and let C_i be the part of C_{i-1} that had 0's in row r (i.e., $C_i \overset{\text{def}}{=} \{c \in C_{i-1} : b_{r,c} = 0\}$). We get stuck if for any i, no row in the current set $[n] \setminus R$ has at least $|C_{i-1}|/2$ 1's in C_{i-1}. Otherwise, we terminate when $C_i = \emptyset$

If we never get stuck, then we generated at most $\log_2(n+1) \leq k$ rows (since $|C_i| \leq |C_{i-1}|/2$, which implies that $|C_{\lceil \log_2(n+1) \rceil}| < 1$). Furthermore, the bit-wise disjunction of these rows is the all 1's row (i.e., for the final R and every $c \in [n]$, it holds that $\vee_{r \in R} b_{r,c} = 1$), since the i^{th} row in R has 1-entries in every column in $C_{i-1} \setminus C_i$, and the last C_i is empty.

On the other hand, if we got stuck at iteration i, then we let $S = C_i$ and note that every row of B has at least $|S|/2$ 0's in the columns S. (This includes the rows in the current R, which each have 0's in all the columns in $S \subset C_{i-1} \subset \cdots \subset C_0$.) In this case, an argument analogous to Adlemam's proof [1] that

$\mathcal{RP} \subseteq \mathcal{P}/\text{poly}$ implies that there exist a set of k columns C that contains a 0-entry in every row (i.e., for every $r \in [n]$, it holds that $\wedge_{c \in C} b_{r,c} = 0$).[6]

Turning to the algorithmics, note that the foregoing procedure for constructing R, S and C 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 C of columns. $\square$

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., $x \notin L$). As mentioned above, by applying Corollary 7, it follows that only one of these equations may hold. Using the decision procedure guaranteed by Corollary 7, we implement Step 3 in our derandomized procedure, and Theorem 2 follows. $\square$

A Finer Analysis. For a $\mathcal{BPP}$ algorithm that 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 it 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 (Theorem 2, refined): *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,*

$$L \in \mathcal{DTIME}(\text{poly}(t(\ell', s \cdot \ell'))),$$

where $\ell' = O(\ell \log \ell)$*.*

We comment that, by using random walks on expanders for error reduction, one can replace $t(\ell', s \cdot \ell')$ by $t(\ell'', s \cdot \ell')$, where $\ell'' = \ell + O(\log \ell) \ll \ell'$.

Acknowledgments. The second author thanks Adam Klivans for explaining [7] to him.

[6] Let us spell out the argument in the current setting. We initiate an iterative process of picking columns from S such that at each iteration we pick a column that covers the largest number of 0's in the remaining rows. That is, we initialize $R_0 = [n]$ and $C = \emptyset$, and for $j = 1, 2, ...$, take a column c not in C that has a maximal number of 0's in R_{i-1}, add c to C, and let R_j be the part of R_{j-1} that has 1's in column c (i.e., $R_j \stackrel{\text{def}}{=} \{r \in R_{i-1} : b_{r,c} = 1\}$). The point is that, by our hypothesis, for the current C, the submatrix $R_{j-1} \times (S \setminus C)$ contains at least $|R_{j-1}| \cdot |S|/2$ 0's, and therefore there exists a column $c \in S \setminus C$ such that $|\{r \in R_{i-1} : b_{r,c} = 0\}| \geq |R_{j-1}|/2$.

References

1. Adleman, L.: Two theorems on random polynomial-time. In: 19th FOCS, pp. 75–83 (1978)
2. Andreev, A.E., Clementi, A.E.F., Rolim, J.D.P.: A new general derandomization method. Journal of the Association for Computing Machinery (J. of ACM) 45(1), 179–213 (1998); Hitting Sets Derandomize BPP. In: XXIII International Colloquium on Algorithms, Logic and Programming, ICALP 1996 (1996)
3. Andreev, A.E., Clementi, A.E.F., Rolim, J.D.P., Trevisan, L.: Weak Random Sources, Hitting Sets, and BPP Simulations. To appear in SICOMP (1997); Preliminary version in 38th FOCS, pp. 264–272 (1997)
4. Buhrman, H., Fortnow, L.: One-sided versus two-sided randomness. In: Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science. LNCS, Springer, Berlin (1999)
5. Even, S., Selman, A.L., Yacobi, Y.: The Complexity of Promise Problems with Applications to Public-Key Cryptography. Inform. and Control 61, 159–173 (1984)
6. Goldreich, O., Zuckerman, D.: Another proof that BPP $\subseteq$ PH (and more). In: Goldreich, O., et al.: Studies in Complexity and Cryptography. LNCS, vol. 6650, pp. 40–53. Springer, Heidelberg (1997)
7. Goldreich, O., Wigderson, A.: Improved derandomization of BPP using a hitting set generator. In: Hochbaum, D.S., Jansen, K., Rolim, J.D.P., Sinclair, A. (eds.) RANDOM 1999 and APPROX 1999. LNCS, vol. 1671, pp. 131–137. Springer, Heidelberg (1999)
8. Impagliazzo, R., Wigderson, A.: P=BPP unless E has Subexponential Circuits: Derandomizing the XOR Lemma. In: 29th STOC, pp. 220–229 (1997)
9. Lautemann, C.: BPP and the Polynomial Hierarchy. Information Processing Letters 17, 215–217 (1983)
10. Sipser, M.: A complexity-theoretic approach to randomness. In: 15th STOC, pp. 330–335 (1983)
11. Zuckerman, D.: Simulating BPP Using a General Weak Random Source. Algorithmica 16, 367–391 (1996)