Electronic Colloquium on Computational Complexity, Report No. 12 (2019)

Multi-pseudodeterministic algorithms[*]

Oded Goldreich[†]

January 27, 2019

To a great woman.

Abstract

In this work, dedicated to Shafi Goldwasser, we consider a relaxation of the notion of pseudodeterministic algorithms, which was put forward by Gat and Goldwasser (*ECCC*, TR11–136, 2011).

Pseudodeterministic algorithms are randomized algorithms that solve search problems by almost always providing the same canonical solution (per each input). Multi-pseudodeterministic algorithms relax the former notion by allowing the algorithms to output one of a bounded number of canonical solutions (per each input). We show that efficient multi-pseudodeterministic algorithms can solve natural problems that are not solveable by efficient pseudodeterministic algorithms, present a composition theorem regarding multi-pseudodeterministic algorithms, and relate them to other known notions.

Keywords: Pseudodeterministic algorithms, search problems, BPP vs P.

Contents

[*]Partially supported by the Israel Science Foundation (grant No. 1146/18).

[†]Faculty of Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel. Email: oded.goldreich@weizmann.ac.il.

ISSN 1433-8092

1 Introduction

Randomized algorithms have a clear deficiancy: They require a source of randomness (or pseudorandomness), and carry a probability of error (or failure). However, as noted by Gat and Goldwasser [1], in the context of solving search problems, randomized algorithms have another deficiancy: They do not necessary return the same answer, when invoked on the same input. At the extreme, which does occur in many natural algorithms, their output is a random variable that has very low (e.g., exponentially small) collision probability.

Randomized algorithms (for solving search problems) that avoid the latter deficiancy (i.e., having a high collision probability) are thus of natural interest. These algorithms offer a functionally that is closer in spirit to that of deterministic algorithms, and are thus called *pseudodeterministic*. That is, pseudodeterministic algorithms are randomized algorithms that solve search problems by almost always providing the same canonical solution.

Definition 1 (pseudodeterministic algorithms [1]): *For a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, let $R(x) \overset{\text{def}}{=} \{y : (x,y) \in R\}$ and $S_R = \{x : R(x) \neq \emptyset\}$. A randomized algorithm A is said to be a* pseudodeterministic solver *of the search problem R if for every $x \in S_R$ there exists* (a canonical solution) *$c_x \in R(x)$ such that* $\Pr[A(x)=c_x] \geq 2/3$ (and $\Pr[A(x)=\perp] \geq 2/3$ for every $x \notin S_R$).

Note that error reduction is applicable to Definition 1; that is, $2/3$ can be replaced by $1-\eta$ if we are willing to repeat the algorithm for $O(\log(1/\eta))$ times.

The notion of pseudodeterministic algorithms was put forward by Gat and Goldwasser [1], who initiated their study. In particular, they presented polynomial-time pseudodeterministic algorithms for several natural search problems, which were previously known to have probabilistic polynomial-time solvers, and characterized the class of search problems having polynomial-time pseudodeterministic algorithms (see Theorem 12). The study of pseudodeterministic algorithms was extended to the sublinear-time model in [4], and to randomized-NC in [5].

We note that it may be that randomization does not help in the context of solving search problem in polynomial-time. This is certainly the case if the promise problem version of $\mathcal{BPP}$ equals the promise version of $\mathcal{P}$ (see Theorem 13). But as long as the latter equality is not known or assumed, pseudodeterministic algorithms are positioned between deterministic and general probabilistic algorithms, and moving from general probabilistic algorithms to pseudodeterministic ones is a step forward, which may even be viewed as a step towards full derandomization. The same holds with respect to the deterministic, pseudodeterministic, and (general) probabilistic versions of $\mathcal{NC}$. In contrast, in the context of sublinear algorithms, probabilistic algorithms are typically significantly stronger than deterministic ones. Unfortunately, the results of [4] indicate that typically pseudodeterministic algorithms are not significantly stronger than deterministic ones.

In light of the above, augmenting the collection of known pseudodeterministic algorithms is of major importance. Having failed to do so, led us to an attempt to relax the notion of pseudodeterministic algorithms, while observing that in the context of sublinear algorithms this relaxation "buys" power.

Indeed, the contents of this work is the presentation of a relaxation of the notion of pseudodeterministic algorithms, and a study of some of its features. The basic notion appears in Section 2.1, and related notions are discussed in Sections 2.2 and 2.3. In Section 3 we present a composition result for this type of algorithms, and in Section 4 we characterize the class of search problems that can be solved by such polynomial-time algorithms in terms of restricted (deterministic polynomail-time) reductions to the promise problem version of $\mathcal{BPP}$.

The computational model. We present several transformations from one type of algorithms to another type, where the latter algorithm invokes the former algorithm as a subroutine (several times). Our transformations presume a model of computation in which the cost (or complexity) of these invocations dominates the complexity of the resulting algorithms. This holds in the standard model of probabilistic (polynomial-time) algorithms as well as in the model of sublinear-time algorithms, but not in the context of log-space computation (e.g., since storing the outcomes of numerous subroutine calls is not for free).

2 Notions

Shafi:	*Do you have the notion of a refill?*
The attendant:	*Yes, we have refills, but what is a notion?*

(In a diner, on the way to STOC'85.)

2.1 The basic notion

One basic difficulty that frustrates attempts to construct pseudodeterministic algorithms is their failure to solve the problem of estimating the average value of a bounded function defined over a huge universe, which is easily solvable by ordinary probabilistic algorithms. The issue is that we can easily find an approximate value, and (w.h.p.) this value can be made to reside in a small interval, but we cannot make it hit a single (canonical) value (with high probability). On the other hand, if we were allowed to hit one of two canonical values (with high probability), then we can easily do it (see Proposition 4). This leads to the following generalization of Definition 1 (which is recovered by setting $m \equiv 1$).

Definition 2 (m-pseudodeterministic algorithm): *For R and S_R as in Definition 1, and $m : \mathbb{N} \to \mathbb{N}$, we say that A is a m-*pseudodeterministic solver* of the search problem R if for every $x \in S_R$ there exists a non-empty set $C_x \subseteq R(x)$ such that $|C_x| \leq m(|x|)$ and $\Pr[A(x) \in C_x] \geq \frac{m(|x|)+1}{m(|x|)+2}$. Furthermore, as in Definition 1, it is required that $\Pr[A(x) = \perp] \geq 2/3$ for every $x \notin S_R$.*

Error reduction is possible here too, since we can distinguish an output not in C_x (which occurs with probability at most $\frac{1}{m+2}$) from at least one of the outputs in C_x (which occurs with probability at least $\frac{1+(1/m)}{m+2}$).[1] In particular, $1/(m+2)$ can be replaced by $\eta < 1/(m+2)$ if we are willing to repeat the algorithm for $O(m^4 \log(1/\eta))$ times. The gap between the probability of outputting some element in $R(x)$ and the probability of outputting any element outside $R(x)$ is a salient feature of Definition 2, as reflected in the following result.

Proposition 3 (key feature of m-pseudodeterministic algorithms): *Let R, S_R and $m : \mathbb{N} \to \mathbb{N}$ be as in Definition 2. Then, any m-pseudodeterministic solver of R, denoted A, satisfies*

1. *For every $x \in S_R$, there exists $c_x \in R(x)$ such that $\Pr[A(x) = c_x] \geq \frac{1+(1/m)}{m+2}$ and for every $y \notin R(x)$ it holds that $\Pr[A(x) = y] \leq 1/(m+2)$.*
2. *For every $x \notin S_R$, it holds that $\Pr[A(x) = \perp] \geq 2/3$.*

On the other hand, if A satisfies the foregoing two conditions, then there exists an m-pseudodeterministic solver of R that invokes A for $\widetilde{O}(m^4)$ times.

Proof Sketch: The necessity follows by an averaging argument (i.e., $\max_{y \in C_x}\{\Pr[A(x) = y]\} \geq \Pr[A(x) \in C_x]/|C_x|$). On the other hand, given A that satisfies the conditions, we consider an algorithm that invokes A for $\widetilde{O}(m^4)$ times and outputs the most frequently occurring output in this sample. Observe that, for any $x \in S_R$, with very high probability, some element of $R(x)$ occurs in the sample with frequency greater than $\frac{1+(1/2m)}{m+2}$ and each element outside of $R(x)$ occurs in the sample with

[1] We use $\frac{m+1}{m+2}$ (resp., $\frac{1}{m+2}$) as a quantity that is noticeably larger than $\frac{m}{m+1}$ (resp., noticeably smaller than $\frac{1}{m+1}$).

frequency smaller than $\frac{1+(1/2m)}{m+2}$. (For proving the last assertion, partition all elements in $\{0,1\}^* \setminus R(x)$ into sets, $S_1, ..., S_{O(m)}$, such that $\Pr[A(x) \in S_i] \leq 1/(m+2)$ for each i, and argue separately for the frequency of the occurrence of each S_i in the sample.) ■

A demonstration of the benefit of the generalization. Indeed, as asserted in the motivating discussion, the generalization from pseudodeterministic algorithms to 2-pseudodeterministic algorithms allows for approximating the average value of a bounded function defined over a huge universe.

Proposition 4 (averages can be approximated by a 2-pseudodeterministic algorithm): *Given oracle access to $f : \{0,1\}^n \to [0,1]$, the value $\overline{f} \stackrel{\rm def}{=} {\rm Exp}_x[f(x)]$ can be approximated, with high probability, to within an additive error of ϵ by a 2-pseudodeterministic algorithm that makes $O(1/\epsilon^2)$ queries to f.*

Recall that (by [4, Thm. 4.1]) any 1-pseudodeterministic algorithm that solves this problem must have query complexity $\exp(\Omega(n))$.

Proof Sketch: The algorithm selects uniformly at random $m = O(1/\epsilon^2)$ points $x_1, ..., x_m \in \{0,1\}^n$, queries f for their value, and outputs $\lfloor v/\epsilon \rceil \cdot \epsilon$ such that $v = \sum_{i \in [m]} f(x_i)/m$, where $\lfloor \alpha \rceil$ is the integer closest to α. The claim follows by noting that, whp, it holds that $|v - \overline{f}| < 0.4\epsilon$. In this case, we have $\lfloor v/\epsilon \rceil \in [\lfloor (\overline{f} - 0.4\epsilon)/\epsilon \rceil, \lfloor (\overline{f} + 0.4\epsilon)/\epsilon \rceil]$, whereas the latter interval contains at most two integers. (Lastly, recall that $|\lfloor v/\epsilon \rceil \cdot \epsilon - v| \leq 0.5\epsilon$).) ■

2.2 An alternative definition

The following definition, which is related to Definition 2, was suggested by Salil Vadhan. It iterprets m-pseudodeterministic algorithms as ones capable of outputing (w.h.p.) a list of m valid solutions that always contains the canonical solution.

Definition 5 (m-pseudodeterministic algorithm, alternative): *For R, S_R and $m : \mathbb{N} \to \mathbb{N}$ as in Definition 2, we say that A is a m-*pseudodeterministic list solver *of the search problem R if for every $x \in S_R$ there exists $c_x \in R(x)$ such that $\Pr[c_x \in A(x) \subseteq R(x)] \geq 1 - \frac{1}{3m(|x|)}$ and $|A(x)| \leq m(|x|)$ always holds. Furthermore, as in Definition 2, it is required that $\Pr[A(x) = \perp] \geq 2/3$ for every $x \notin S_R$.*

The choice of the probability bound in the main condition of Definition 5 is more arbitrary than in Definition 2. For starters, as in the latter case, any lower bound that is noticeablly larger than $m/(m+1)$ (and smaller than $1 - \exp(-m)$) will do (see proof of Theorem 6).[2] Furthermore, even an lower bound that is (only) noticeablly larger than $1/2$ allows for equivalence with Definition 2, albeit at the cost increasing the parameter m. (For example, a lower bound of $2/3$ in the main condition allows for an equivalence up to a constant factor in the parameter m; see the proof of Theorem 6.)[3]

Theorem 6 (relating Definitions 2 and 5): *For every efficiently computable $m : \mathbb{N} \to \mathbb{N} \setminus \{1\}$ and search problem R, the following hold.*

1. *If R has an m-pseudodeterministic solver, then, invoking this solver for $\widetilde{O}(m^4)$ times yields a m-pseudodeterministic list solver.*

[2]Specifically, note the free parameter η in the proof of the first direction, and observe that in the opposite direction any lower bound p such that $p > m/(m+1)$ suffices in order to assert that $c_x \in \{y : q_y > 1 - 1/(m+1)\} \subseteq R(x)$. Hence, the choice of p only affects the overhead $(p - (m/(m+1)))^2$.

[3]Specifically, in such a case, we shall use the fact that $q_{c_x} \geq 2/3$, whereas $|\{y : q_y > 1/2\}| < 2m$ and $q_y < 1/2$ for every $y \notin R(x)$. Outputting an arbitrary element that appears in at least 60% of the lists, we derive an $2m$-pseudodeterministic solver. In general, any lower bound p such that $p > 1/2$ implies $|\{y : q_y > 1/2\}| < 2m$, whereas the choice of $p > 1/2$ only affects the overhead $(p - 0.5)^2$.

2. *If R has an m-pseudodeterministic list solver, then, invoking this solver for $\widetilde{O}(m^2)$ times yields a m-pseudodeterministic solver.*

Proof: Let A be an m-pseudodeterministic solver of R. For an arbitrary function $\eta : \mathbb{N} \to (0, 1/3]$, consider an algorithm that on input x proceeds as follows.

1. Invokes $A(x)$ for $O(m(|x|)^4 \cdot \log(1/\eta(|x|)))$ times, and let p_y denote the frequency of the output y in these invocations.

2. If $p_\perp > 1/2$, then we output $\perp$. Otherwise, we output the list of all y's such that $p_y > \frac{1+(1/2m(|x|))}{m(|x|)+2}$.

Clearly, if $x \notin S_R$, then we output $\perp$ with very high probability. Turning to the case of $x \in S_R$, let C_x be as in Definition 2, and let $c_x \in C_x$ be a string that is output by $A(x)$ with the highest probability (with ties broken arbitrarily).[4] Then, $\Pr[A(x) = c_x] \geq \frac{1+(1/m(|x|))}{m(|x|)+2}$, and so, with probability at least $1-\eta(|x|)/2$, it holds that $p_{c_x} > \frac{1+(1/2m(|x|))}{m(|x|)+2}$. On the other hand, $\Pr[A(x) \notin C_x] \leq \frac{1}{m(|x|)+2}$, and so, with probability at least $1 - \eta(|x|)/2$, it holds that $\sum_{y \notin C_x} p_y < \frac{1+(1/2m(|x|))}{m(|x|)+2}$. Recalling that $|C_x| \leq m(|x|)$ and $C_x \subseteq R(x)$, it follows that, with probability at least $1 - \eta(|x|)$, we output a list of size at most $m(|x|)$ that contain c_x and is contained in $R(x)$. Setting $\eta(n) = 1/3m(n)$, Part 1 follows (since we may output $\perp$ in the rare case that the aforementioned list exceeds the size bound).

Turning to Part 2, let A be an m-pseudodeterministic list solver of R, and consider the probability, denoted q_y, that y occurs in the list output by $A(x)$; that is, $q_y \stackrel{\text{def}}{=} \Pr[y \in A(x)]$. Then, $\sum_y q_y \leq m(|x|)$, and so $|\{y : q_y > 1-1/(m(|x|)+1)\}| \leq m(|x|)$. Also note that $q_{c_x} \geq 1-1/3m(|x|) > 1-1/(m(|x|)+1)$, and that $q_y \leq 1/3m(|x|) < 1 - 1/(m(|x|) + 1)$ for every $y \notin R(x)$. Now, consider an algorithm that on input x proceeds as follows.

1. Invokes $A(x)$ for $O(m(|x|)^2 \cdot \log m(|x|))$ times, and let p_y denote the frequency of y in the lists produces in these invocations.

2. If some y satisfies $p_y > 1 - 1/2m(|x|)$, then we output it (i.e., if several y's satisfy $p_y > 1 - 1/2m(|x|)$, then we output one of them chosen arbitrarily). Otherwise, we output $\perp$.

Clearly, if $x \notin S_R$, then we output $\perp$ with very high probability. Otherwise (i.e., $x \in S_R$), with probability at least $1-1/2(m(|x|)+2)$, the solution c_x occurs more than a $1-1/2m(|x|)$ fraction of the lists output in the invocations of $A(x)$ (since $q_{c_x} \geq 1 - 1/3m(|x|)$). Likewise, with probability at least $1 - 1/2(m(|x|) + 2)$, no y such that $q_y \leq 1 - 1/(m(|x|) + 1) < 1 - 3/5m(|x|)$ occurs in a $1 - 1/2m(|x|)$ fraction of these lists.[5] ■

2.3 Relation to reproducible solution algorithms

The following (general) notion of reproducible solution algorithms is implicit in the work of Grossman and Liu [6], which focuses on the case of randomized log-space.

Definition 7 (t-reproducible algorithms): *For R, S_R as in Definition 2 and $t : \mathbb{N} \to \mathbb{N}$, we say that A is a t-*reproducible solver *of the search problem R if A decomposes to two algorithms, A_1 and A_2, such that the following conditions hold.*

1. *$|A_1(x)| = t(|x|)$ always holds, and for $x \notin S_R$ it holds that $\Pr[A_2(x, A_1(x)) = \perp] \geq 2/3$.*

[4] Indeed, we could have used the first part of Proposition 3 instead (for arguing that some element c_x of $R(x)$ satisfies $\Pr[A(x) = c_x] \geq \frac{1+(1/m(|x|))}{m(|x|)+2}$), but not for arguing that $\sum_{y \notin R(x)} \Pr[A(x) = y]$ is small.

[5] Again, this is shown by partitioning all these y's into sets, $S_1, ..., S_{O(m)}$, such that $\Pr[A(x) \in S_i] \leq 1 - 1/(m + 1)$ for each i, and argue separately for the frequency of each S_i in the sample.

2. *For every $x \in S_R$ and every y, there exists $c_{x,y} \in R(x)$ such that*

$$\mathrm{Exp}_{y \leftarrow A_1(x)} \left[\Pr[A_2(x,y) = c_{x,y}]\right] \geq 8/9.$$

Algorithm A_2 (and consequently algorithm A) *is* sound *if for every $x \in S_R$ and every y, it holds that* $\Pr[A_2(x,y) \in R(x) \cup \{\perp\}] = 1$. *Algorithm A is* almost-sound *if, for every $x \in S_R$, it holds that* $\Pr[A_2(x, A_1(x)) \in R(x) \cup \{\perp\}] = 1 - 2^{-4t(|x|)-1}$.

We say that y is x-good if $\Pr[A_2(x,y) = c_{x,y}] \geq 2/3$, and note that Condition 2 implies that with probability at least 2/3 it holds that $A_1(x)$ is x-good. Observe that, like in Definition 2, the error probability of algorithm A_2 on inputs (x,y) such that y is x-good can be reduced by repetitions. When A_2 is sound, the error probability of algorithm A_1 can also be reduced by repetitions (and testing the potential outputs by invoking A_2).[6] This fact is used in Part 1 of the following result.

Theorem 8 (relating Definitions 2 and 5): *For every efficiently computable $t : \mathbb{N} \to \mathbb{N}$ and search problem R, the following hold.*

1. *If R has a t-reproducible almost-sound solver, then, invoking this solver for $O(t^2)$ times yields a 2^t-pseudodeterministic solver.*

2. *If R has an 2^t-pseudodeterministic solver, then, invoking this solver for $\exp(O(t))$ times yields a $(t+4)$-reproducible almost-sound solver.*

Part 2 was suggested to us by Ofer Grossman.

Proof: Starting with Part 1 and employing error reduction, we obtain algorithms A'_1 and A'_2 such that for every $x \in S_R$

$$\Pr_{y \leftarrow A'_1(x)} \left[\Pr[A'_2(x,y) = c_{x,y}] > 1 - 2^{-2t(|x|)-1}\right] \geq 1 - 2^{-2t(|x|)-1}$$

where $c_{x,y} \in R(x)$ is as in Definition 7. Specifically, $A'_1(x)$ invokes $A_1(x)$ for $2t(|x))$ times, and for each sampled output y it invokes $A_2(x,y)$ for $O(t(|x|))$ times, thus estimating the collision probability of $A_2(x,y)$ up to additive deviation of 0.1 with probability at least $1 - 2^{-2t(|x|)-2}$. Furthermore, with probability at least $1 - 2^{-2t(|x|)-2}$, for one of the sampled y's the collision probability of $A_2(x,y)$ is at least 2/3, whereas all sampled y's are in $R(x) \cup \{\perp\}$ (since A is almost-sound). Likewise, $A'_2(x,y)$ invokes $A_2(x,y)$ for $O(t(|x))$ times, and outputs the most frequently occurring output.

Letting $C_x = \{c_{x,y} : y \in \{0,1\}^{t(|x|)}\} \cap R(x)$, we consider an algorithm that on input x outputs $A'_2(x, A'_1(x))$. Observe that on input $x \in S_R$, this algorithm outputs an element of C_x with probability at least $(1 - 2^{-2t(|x|)-1})^2 > 1 - 1/(2^{t(|x|)} + 2)$. The claim follows (noting that $|C_x| \leq 2^{t(|x|)}$ and that the algorithm outputs $\perp$ w.h.p whenever $x \notin S_R$).

Turning to Part 2, we are given an algorithm A that on input $x \in S_R$ satisfies $\Pr[A(x) \in C_x] \geq \frac{m+1}{m+2}$, where $m = 2^{t(|x|)}$. The basic idea is distinguishing between elements in a non-empty subset of C_x that each occur in $A(x)$ with probability at least $\frac{1+(1/2m)}{m+2}$ and the other elements.[7] The problem is that some elements of C_x may occur with probability that is very close to this threshold. This problem is resolved by using a random threshold in the interval $\left[\frac{1+(1/2m)}{m+2} \pm \frac{1/3m}{m+2}\right]$. Specifically, we select the threshold at random in the set $T = \left\{\frac{1+(1/6m)}{m+2} + \frac{i/15m^2}{m+2} : i \in [10m]\right\}$.

[6]Specifically, we sample a few outputs y's of $A_1(x)$, and output the one that seems to maximize the collision probability of $A_2(x,y)$. That is, for each sampled y, we estimate the probability that two invocations of $A_2(x,y)$ yield the same output, by making a small number of trials.

[7]Recall that (as shown in Proposition 3) some string in C_x appears in $A(x)$ with probability at least $\frac{1+(1/m)}{m+2}$, whereas each string outside C_x appears in $A(x)$ with probability at most $\frac{1)}{m+2}$.

Using the fact that the elements in T are at distance at least $\frac{1}{15m^2\cdot(m+2)}$ apart, it follows that each element in $\{\Pr[A(x)=y] : y \in C_x\}$ can be at distance at least $\frac{1}{31m^2\cdot(m+2)}$ from at most one of the elements of T. Hence, with probability at least 0.9, the random threshold τ is at distance at least $\frac{1}{31m^2\cdot(m+2)}$ from each of the m probabilities of occurrence of elements in C_x (and is always at distance at least $1/(6m\cdot(m+2))$ from the probability of occurrence of any element outside of C_x). Thus, we obtain the following $(t+4)$-reproducible solver: The first component (i.e., A'_1) outputs a uniformly distributed $i \in [10\cdot 2^{t(|x|)}]$, and the second component (i.e., A'_2) uses i to determine the threshold $\tau = \frac{1+((5\cdot 2^{t(|x|)-1}+i)/15\cdot 2^{2t(|x|)})}{2^{t(|x|)}+2}$ and the set $C'_{x,\tau} = \{y : \Pr[A(x)=y] > \tau\}$. Whenever the threshold is good (in the above sense), the set $C'_{x,\tau}$ is correctly reconstructed (with probability $1-o(1)$, by using $\widetilde{O}(2^{6t(|x|)})$ invocations of $A(x)$), and $A'_2(x,i)$ may just output the lex-first element of $C'_{x,\tau}$. Finally, observe that (A'_1, A'_2) is almost-sound, because for every τ, with probability at least $1-2^{-4t(|x|)-1}$, the set $C'_{x,\tau} \subset R(x)\cup\{\perp\}$ is correctly reconstructed. ■

3 Multiple invocations of multi-pseudodeterministic algorithms

A straightforward application of Proposition 4 to the problem of approximating t different quantities yields a 2^t-pseudodeterministic algorithm. We can do better.

Algorithm 9 (approximating t averages by a $(t+1)$-pseudodeterministic algorithm): *Given oracle access to $f_1, ..., f_t : \{0,1\}^n \to [0,1]$, the algorithm proceeds as follows.*

1. *It selects uniformly at random $m = \widetilde{O}(t^4)/\epsilon^2$ points, $x_1, ..., x_m \in \{0,1\}^n$, queries the f_i's for their value, and computes $v_i = \sum_{j\in[m]} f(x_j)/m$ for each $i \in [t]$.*
2. *It selects uniformly at random a number $\tau \in \{(j-0.5)\cdot\epsilon/10t^2 : j \in [5t^2]\}$, and sets $v'_i = v_i + \tau$ for each $i \in [t]$.*

It outputs the t-tuple $(\lfloor v'_1/\epsilon\rceil\cdot\epsilon, ..., \lfloor v'_t/\epsilon\rceil\cdot\epsilon)$.

Indeed, with probability at least $1-\frac{1}{2(t+3)}$, it holds that $|v_i - \overline{f}_i| < \epsilon/10t^2$ for every $i \in [t]$, where $\overline{f}_i \overset{\text{def}}{=} \text{Exp}_x[f_i(x)]$, and in this case $\lfloor(v_i+\tau)/\epsilon\rceil \in I_i$, where $I_i = [\lfloor(\overline{f}_i+\tau-\epsilon/10t^2)/\epsilon\rceil, \lfloor(\overline{f}_i+\tau+\epsilon/10t^2)/\epsilon\rceil]$. We call τ bad for i if the interval I_i contains more than a single integer, and note that the probability that τ is bad for some $i \in [t]$ is at most

$$t\cdot\Pr_\tau[\lfloor(\overline{f}_i+\tau-\epsilon/10t^2)/\epsilon\rceil \neq \lfloor(\overline{f}_i+\tau+\epsilon/10t^2)/\epsilon\rceil],$$

which is upper-bounded by $t\cdot 4/10t^2 < 1/2(t+3)$ (assuming $t > 12$). Hence, with probability $1-1/(t+3)$, all v_i's are $\epsilon/10t^2$-close to the corresponding $\overline{f}_i$'s and τ is not bad for any i. In this case, the output sequence equals $(\lfloor(\overline{f}_1+\tau)/\epsilon\rceil\cdot\epsilon, ..., \lfloor(\overline{f}_t+\tau)/\epsilon\rceil\cdot\epsilon)$, and the key observation is that this sequence can assume at most $t+1$ possible values when τ varies in the interval $[0,\epsilon/2]$. To see this, let $\text{rem}_\epsilon(\alpha) = \alpha - \lfloor\alpha/\epsilon\rfloor\cdot\epsilon$, and assume (w.l.o.g.) that $\text{rem}_\epsilon(\overline{f}_i) \leq \text{rem}_\epsilon(\overline{f}_{i+1})$ for every $i \in [t-1]$. In this case a typical sequence $(\lfloor(\overline{f}_1+\tau)/\epsilon\rceil\cdot\epsilon, ..., \lfloor(\overline{f}_t+\tau)/\epsilon\rceil\cdot\epsilon)$ takes the floor-value on the first $i \in \{0,1,...,t\}$ values and the ceiling-value in the remaining $t-i$ values, which means that there are at most $t+1$ possibilities. It follows that *Algorithm 9 constitute a $(t+1)$-pseudodeterministic algorithm for approximating the averages of t bounded functions.*

Comment. Algorithm 9 is closely related to the $O(t\log(1/\epsilon))$-reproducible solver used by Grossman and Liu [6] to estimate $\exp(O(t))$ different probabilities (which arise from any randomized 2^t-time algorithm that uses space t). Using Part 1 of Theorem 8, this yields a $\text{poly}(t/\epsilon)$-pseudodeterministic algorithm, whereas our analysis yields a $(t+1)$-pseudodeterministic algorithm. The use of a single threshold here (and in [6]) is reminiscent of the use of a single threshold by Saks and Zhou [7], although the benefit of using a single threshold is different. Specifically, Saks and Zhou [7] use a single threshold in order to economize on randomness, whereas we use it in order to better upper-bound the number of possible rounding-patterns (which would have been 2^t otherwise).

Beyond Algorithm 9. Generalizing the idea that underlies the analysis of Algorithm 9, we obtain the following composition result (which improves over the trivial bound of m^t).

Theorem 10 (non-adaptive invocations of a m-pseudodeterministic algorithm): *Suppose that A is an m-pseudodeterministic algorithm for solving the search problem R. Then, one can solve t instances of R by a $(t\cdot(m-1)+1)$-pseudodeterministic algorithm that invokes A for* $\text{poly}(tm)$ *times.*

Note that this result holds both in the context of sub-linear time algorithms (as in Algorithm 9) and in the context of polynomial-time algorithms.

Proof: Let $(x_1,...,x_t)$ be a sequence of inputs, and suppose for simplicity that $x_1,...,x_t \in S_R$ (since the x_i's that have no solution are easy to detect).

For every $x \in S_R$, let $p_y(x) = \Pr[A(x)=y]$, and observe that $\max_y\{p_y(x)\} \geq \frac{1}{m}\cdot\frac{m+1}{m+2} = (1+m^{-1})\cdot\frac{1}{m+2}$ whereas $\{y : p_y(x) > 1/(m+2)\} \subseteq C_x$ (where $C_x \subseteq R(x)$ is as defined in Definition 2). Letting $\delta = 1/(m\cdot(m+2))$, this suggests finding, for each $x \in S_R$, the set of y's such that $p_y(x) \geq f(\tau) \overset{\text{def}}{=} \frac{1}{m+2}+\tau$, where τ is uniformly distributed in $[0.5\delta,\delta]$. Specifically:

1. Select τ uniformly in $[0.5\delta,\delta]$. Indeed, selecting τ in $\{(M+i)\cdot\delta/2M : i \in [M]\}$, where $M = \text{poly}(tm)$, will do.

2. For each $i \in [t]$, taking a sample of size $\text{poly}(M/\epsilon)$, let C_i denote the set of y's that appeared with frequency at least $f(\tau)$ in the sample.

3. For each $i \in [t]$, output the lex-first element of C_i.

With very high probability, each C_i is non-empty and contains only elements of C_{x_i}. Furthermore, each $y \in C_i$ occurs in the sample with frequency that is very close to $p_y(x_i)$. We call τ bad for i if for some y such that $p_y(x_i) > f(0) = 1/(m+2)$ it holds that $|p_y(x_i) - f(\tau)| < \delta/4M$, and observe that the probability that τ is bad for some i is $o(1/tm)$. We complete the proof by showing that the algorithm's output, when τ is not bad for any i, is one of $t\cdot(m-1)+1$ possibilities.

To see this, fix such a τ, and consider the set $\{(i,y) : p_y(x_i) > f(0)\}$. Arrange all these $t\cdot m$ pairs according to the value of $p_y(x_i)$, and note that $f(\tau)$ is sufficiently far from each of these values. Hence, with probability $1-o(1/tm)$, each set C_i equals the set of y's such that $p_y(x_i) > f(\tau)$. It follows that, in this case, the output of the algorithm is uniquely determined by the set $\{(i,y) : p_y(x_i) > f(\tau)\}$, or, equivalently, by the location of τ in the sorted sequence of $p_y(x_i)$'s. Note that there are only $tm+1-t$ (rather than $tm+1$) possibilities, since for each i there exists y such that $p_y(x_i) > f(\delta)$. ■

Adaptive invocations. We note that a result of the flavor of Theorem 10 for adaptive invocations of 2-pseudodeterministic algorithms (i.e., invoking the algorithm on instances determined by prior invocations) would imply efficient poly-pseudodeterministic algorithms for all "search problems in BPP" (as defined in [3, Sec. 3.1]). This is the case because "BPP search problems" are deterministically reducible (in polynomial-time) to promise problems in BPP (see [3, Thm. 3.5]), which in turn have polynomial-time 2-pseudodeterministic algorithms.

4 A complexity theoretic perspective

Confining ourselves to search problems having "efficiently recognizable solutions" (as defined next), we characterize the class of search problems that are solavle by m-pseudodeterministic algorithms in terms of problems that are solvable by deterministic polynomial-time reductions of a certain type.

Definition 11 (search problems with efficiently recognizable solutions): *A search problem $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is said to have* efficiently recognizable solutions *if there exists a probabilistic polynomial time algorithm for deciding membership in R* (i.e., R, as a set, is in $\mathcal{BPP}$).

Our starting point is the charcterization provided by Gal and Golkdwasser [1].

Theorem 12 (characterization of the class of search problems that are solvable by polynomial-time pseudodeterministic algorithms [1]):[8] *Let R be a search problem having efficiently recognizable solutions. The problem R can be solved by a polynomial-time pseudodeterministic algorithm if and only if it is reducible in determinstic polynomial-time to a decision problem in $\mathcal{BPP}$.*

Indeed, the reductions referred to in Theorem 12 are deterministic polynomial-time oracle machines that make multiple queries to the (binary) oracle, which in turn is (the characteristic function of) a set in $\mathcal{BPP}$. Interestingly, using reductions to the promise problem version of $\mathcal{BPP}$, denoted pr$\mathcal{BPP}$, allows such reductions to solve any "BPP search problem" (i.e., a search problem that can be solved in probabilistic polynomial-time and has efficiently recognizable solutions).

Recall that a promise problem is a pair of disjoint sets, $(S_{\rm YES}, S_{\rm NO})$, and solving it means distinguishing inputs in $S_{\rm YES}$ from inputs in $S_{\rm NO}$; the set $S_{\rm YES} \cup S_{\rm NO}$ is called the promise. Consequently, when a reduction to a promise problem makes a query that lies outside the promise (or "violates the promise"), it obtains an arbitrary answer (see [2, Sec. 2.4.1.1]).

Theorem 13 (characterization of the class of search problems that are solvable by probabilistic polynomial-time algorithms [3]): *Let R be a search problem having efficiently recognizable solutions. The problem R can be solved by a* (two-sided error) *probabilistic polynomial-time algorithm if and only if it is reducible in deterministic polynomial-time to a promise problem in* pr$\mathcal{BPP}$.

It turns out that Theorems 12 and 13 are special cases of a general result, which refers to the number of queries outside the promise that appear in the "tree of all possible executions" of the reduction. The latter notion is defined next.

Definition 14 (the directed tree of all possible executions of a reduction): *Let M be deterministic reduction to a promise problem $\Pi = (S_{\rm YES}, S_{\rm NO})$; that is, M is a deterministic oracle machine that makes oracle calls to Π. For any input x, the* tree of all possible executions of *$M^\Pi(x)$ describes all possible executions of M on input x and oracle Π such that internal nodes in this directed tree are associated with queries that may be made in such an execution, outgoing edges in the tree correspond to possible answers to these queries, and leaves correspond to the outputs in such executions. Specifically, a query q that satisfies the promise of Π has a single child in the tree, since the answer to q is 1 if $q \in S_{\rm YES}$ and 0 if $q \in S_{\rm NO}$, whereas a query that violates the promise has two children corresponding to the two possible answers to the query $q \notin S_{\rm YES} \cup S_{\rm NO}$.*

Note that (standard) decision problems have trivial promises (i.e., the decision problem associated with the set S corresponds to the promise problem $(S, \{0,1\}^* \setminus S)$), and so the tree of all possible executions of a (deterministic) reduction to a decision problem consists of a single path (i.e., all internal nodes have

[8]Actually, the equivalence holds (and is stated in [1]) also for search problems not having efficiently recognizable solutions.

degree 1 and the directed tree has a single leaf). On the other hand, the tree of all possible executions of a deterministic polynomial-time reduction to a promise problem may contain an exponential number of internal nodes.

Theorem 15 (characterization of the class of search problems that are solvable by polynomial-time m-pseudodeterministic algorithms):[9] *Let R be a search problem having efficiently recognizable solutions, and $m : \mathbb{N} \to \mathbb{N}$ be upper-bounded by a polynomial. The problem R can be solved by a polynomial-time m-pseudodeterministic algorithm if and only if it is reducible in determinstic polynomial-time to a promise problem Π in* pr$\mathcal{BPP}$ *such that on input x the tree of all possible executions of $M^\Pi(x)$ contains at most $m(|x|)$ leaves.*

Note that Theorem 12 is a special case of Theorem 15 in which $m \equiv 1$, whereas Theorem 13 can be restated by replacing the two occurrences of m (in the main assertion of Theorem 15) by two different functions of the form exp(poly).[10]

Proof: Suppose that R can be solve by a deterministic (polynomial-time) oracle machine M when given oracle access to a promise problem $\Pi \in \text{pr}\mathcal{BPP}$ such that the tree of all possible executions of $M^\Pi(x)$ contains at most $m = m(|x|)$ leaves. Then, this tree contains at most $m-1$ internal nodes that correspond to queries that lie outside the promise of Π. The idea is to emulate a possible execution of $M^\Pi(x)$ by replacing the oracle calls with an actual probabilistic polynomial-time computation. When the query lies inside the promise, doing so is straightforward (since by the hypothesis $\Pi \in \text{pr}\mathcal{BPP}$), and the answer is uniquely determined and is correctly obtained (with overwhelmingly high probability). However, when the query lies outside the promise, we can estimate the acceptance probability of the machine deciding Π, and decide according to a single randomly selected threshold (as in the proof of Theorem 10).

Actually, we (can not and) do not need to know whether or not the query lies inside the promise; we rather estimate the acceptance probability of the machine deciding Π, and act accordingly (while relying on the fact that if the query lies inside the promise then its acceptance probability is bounded away from the selected threshold). Before detailing the resulting algorithm, we observe that all queries in the tree (including the $m-1$ queries that violate the promise) are fully determined by M, x and Π, which means that our decision is determined whenever the random threshold is sufficiently far from the accepting probabilities of all queries (including the $m-1$ queries that violate the promise). As in the proof of Theorem 10, it follows that the resulting (polynomial-time) algorithm is a $((m-1)+1)$-pseudodeterministic algorithm that solves R. Specifically, using a decision procedure P for Π, on input x, our randomized algorithm proceeds as follows.

1. Select $\tau \in \left\{0.4 + \frac{i}{5(m+2)^2} : i \in [(m+2)^2]\right\}$ uniformly at random.
2. Emulate the execution of M on input x, answering queries by estimating the acceptance probability of P and returning 1 if and only if the estimate is larger than τ. Specifically, M makes a query q, we estimate $p_q \overset{\text{def}}{=} \Pr[P(q)=1]$ such that, with probability at least $1-(m+2)^{-2}/d$, the estimate $\widetilde{p}_q$ falls inside $[p_q \pm 0.09 \cdot (m+2)^{-2}]$, where d is the depth of the tree of all possible executions of $M^\Pi(x)$. We return 1 if $\widetilde{p}_q > \tau$ and 0 otherwise.

[9]Actually, the equivalence holds also for search problems not having efficiently recognizable solutions; indeed, the following proof does not refer to this hypothesis.

[10]The resulting assertion reads: *For any search problem R having efficiently recognizable solutions, R can be solved by a polynomial-time* exp(poly)*-pseudodeterministic algorithm if and only if it is reducible in deterministic polynomial-time to a promise problem Π in* pr$\mathcal{BPP}$ *such that on input x the tree of all possible executions of $M^\Pi(x)$ contains at most* exp(poly($|x|$)) *leaves.* The proof uses the fact that if R has efficiently recognizable solutions, then any probabilistic polynomial-time algorithm solving R can be made to output valid solutions for x with probability at least $1-\exp(-\text{poly}(|x|))$.

Note that providing the aforementioned approximation requires $O(m^4 \log(md)) \leq \text{poly}(|x|)$ invocations of P, where the inequality uses the upper bounds on m and on the running time of M.

3. When M halts, we output its verdict.

Observe that τ is very far from any query that lies inside the promise of Π (i.e., for every such query q it holds that either $p_q \geq 2/3$ or $p_q \leq 1/3$). As for queries that lie outside the promise of Π, the probability that any of these queries is $0.09 \cdot (m+2)^{-2}$-close to τ is at most $m \cdot (m+2)^{-2} < (m+2)^{-1} - (m+2)^{-2}$. Furthermore, when all queries are $0.09 \cdot (m+2)^{-2}$-far from τ, with probability $1-(m+2)^{-2}$ the relation of each of the $\widetilde{p}_q$'s to τ is determined (by the relation of $t+p$ to τ), and the claim follows as in the proof of Theorem 10.

Turning to the opposite direction, let A be an m-pseudodeterministic algorithm that solves R. The oracle machine that we construct, iteratively extends a prefix of a single solution that is output by $A(x)$ with probability at least $\frac{1+(1/m(|x|))}{m(|x|)+2}$, provided $x \in S_R$. Indeed, in each iteration the current prefix is extended by a single bit such that the probability of outputting a single solution that extends this prefix is essentially preserved (or even increased). In order to determine this bit, we need to estimate the probability that $A(x)$ outputs a single solution that fits the extended prefix, which can be done by issuing an adequate query to pr$\mathcal{BPP}$.

Note that we cannot hope distinguish in pr$\mathcal{BPP}$ between the case that the extended prefix fits a single solution that is output with probability at least p and the case that each fitting solution is output with probability smaller than p, but we can distinguish (in pr$\mathcal{BPP}$) between the former case and the case that the latter probability is smaller than $p - 1/\text{poly}(|x|)$. Hence, in the i^{th} iteration we seek a i-bit long prefix that fits some single solution that is output by $A(x)$ with probability at least $\frac{1+(1/m(|x|)-(i/\text{poly}(|x|))}{m(|x|)+2}$. We enter this iteration with a $(i-1)$-bit long prefix y of a single solution that is output by $A(x)$ with probability at least $\frac{1+(1/m(|x|)-((i-1)/\text{poly}(|x|))}{m(|x|)+2}$, and use the oracle in order to distinguish the case that $y0$ is output by $A(x)$ with probability at least $\frac{1+(1/m(|x|)-((i-1)/\text{poly}(|x|))}{m(|x|)+2}$ and the case that $y0$ is output by $A(x)$ with probability smaller than $\frac{1+(1/m(|x|)-(i/\text{poly}(|x|))}{m(|x|)+2}$.

The key observation is that this query (i.e. $y0$) violates the corresponding promise if and only if both possible setting of the current bit yield a prefix that can be extended to a single solution that is output by $A(x)$ with about the same (or higher) probability. (Specifically, if $(x, y0)$ is not a YES-instance, then $y1$ must be the extension of y that fits a solution output by $A(x)$ with probability at least $\frac{1+(1/m(|x|)-((i-1)/\text{poly}(|x|))}{m(|x|)+2}$.) It follows that the number of leaves in the execution tree of the oracle machine that we construct cannot exceed the number of solutions that $A(x)$ outputs with probability greater than $1/(m(|x|)+2)$, which is upper-bounded by $m(|x|)$. The resulting oracle machine, denoted M, is detailed next, while assuming (without loss of generality)[11] that all solutions in $R(x)$ have length $\ell(|x|)$, for some easy to compute function $\ell : \mathbb{N} \to \mathbb{N}$.

1. On input x, estimate $\Pr[A(x) = \perp]$, and halt outputting $\perp$ if the estimate exceeds 1/2. (This estimate is obtained by making the query x to the $\mathcal{BPP}$-set $\{x' : \Pr[A(x') = \perp] \geq 2/3\} = \{0,1\}^* \setminus S_R$, while noting that $\Pr[A(x) = \perp] \leq 1/3$ for any $x \in S_R$.)

   Otherwise (i.e,., $x \in S_R$), the machine initializes $y \leftarrow \lambda$ as a prefix of some solution that is output with sufficiently high probability. Indeed, at this point, there exists $z \in \{0,1\}^{\ell(|x|)-|y|}$ such that $\Pr[A(x) = yz] \geq \frac{1+(1/m(|x|))}{m(|x|)+2}$.

   Let $\delta(n) = 1/m(n)$ and $\epsilon(n) = 1/(2m(n) \cdot \ell(n))$.

[11] We may set ℓ to be the running time of A, and consider padding all solutions to length $\ell(|x|)+1$; for example, consider $R'(x) \overset{\text{def}}{=} \{y10^{\ell(|x|)-|y|} : y \in R(x)\}$.

2. For $i = 1, ..., \ell(|x|)$, test if extending the prefix y by 0 yields a prefix of a single solution that is output with sufficiently high probability. Specifically, we estimate the probability that $A(x)$ outputs a single solution with prefix $y0$, set $y \leftarrow y0$ if the estimate exceeds $\frac{1+\delta(|x|)-(i-1)\cdot\epsilon(|x|)}{m(|x|)+2}$, and set $y \leftarrow y1$ otherwise.

   The foregoing estimate is obtained by making the query $(x, y0)$ to the promise problem $\Pi = (S_{\rm YES}, S_{\rm NO})$ such that

$$\begin{aligned} S_{\rm YES} &= \left\{(x', y') : \exists z \in \{0,1\}^{\ell(|x'|)-|y'|} \text{ s.t. } \Pr[A(x') = y'z] \geq \frac{1+\delta(|x'|)-(|y'|-1)\cdot\epsilon(|x'|)}{m(|x'|)+2}\right\} \\ S_{\rm NO} &= \left\{(x', y') : \forall z \in \{0,1\}^{\ell(|x'|)-|y'|} \ \ \Pr[A(x') = y'z] < \frac{1+\delta(|x'|)-|y'|\cdot\epsilon(|x'|)}{m(|x'|)+2}\right\} \end{aligned}$$

   That is, we extend $y \in \{0,1\}^{i-1}$ to $y0$ if and only if the query $(x, y0)$ is answered positively. Hence, if $y \in \{0,1\}^{i-1}$ is extended to $y0$, then $(x, y0) \notin S_{\rm NO}$ must hold, which implies that $A(x)$ outputs a single solution with prefix $y0$ with probability at least $\frac{1+\delta(|x|)-i\cdot\epsilon(|x|)}{m(|x|)+2}$. On the other hand, if $y \in \{0,1\}^{i-1}$ is extended to $y1$, then $(x, y0) \notin S_{\rm YES}$ must hold, which implies that $A(x)$ outputs a single solution with prefix $y1$ with probability at least $\frac{1+\delta(|x|)-(i-1)\cdot\epsilon(|x|)}{m(|x|)+2}$, since otherwise $A(x)$ outputs each single solution with prefix y with probability less than $\frac{1+\delta(|x|)-(i-1)\cdot\epsilon(|x|)}{m(|x|)+2}$, which contradicts the guarantee of the prior iteration. In both cases, the extended i-bit long y satisfies the following: there exists $z \in \{0,1\}^{\ell(|x|)-i}$ such that $\Pr[A(x) = yz] \geq \frac{1+\delta(|x|)-i\cdot\epsilon(|x|)}{m(|x|)+2}$.

   Note that the promise problem Π is in pr$\mathcal{BPP}$ by virtue of an algorithm that, on input (x', y'), invokes $A(x')$ for $O(m(|x'|)/\epsilon(|x'|))^2 = \text{poly}(|x'|)$ times and estimate the maximal probability that the output equals a single string that extends y'.

3. Output $y \in \{0,1\}^{\ell(|x|)}$, while noting that

$$\Pr[A(x) = y] \geq \frac{1+\delta(|x|)-|y|\cdot\epsilon(|x'|)}{m(|x|)+2} = \frac{1+(1/2m(|x|))}{m(|x|)+2}$$

Note that whenever the query $(x, y0)$ lies outside $S_{\rm YES} \cup S_{\rm NO}$ it is the case that for both $\sigma \in \{0,1\}$ it holds that $A(x)$ outputs a single solution with prefix $y\sigma$ with probability at least $\frac{1+\delta(|x|)-|y\sigma|\cdot\epsilon(|x|)}{m(|x|)+2}$, which equals $\frac{1+(1/m(|x|))-(|y\sigma|/2m(|x|)\ell(|x|))}{m(|x|)+2} \geq \frac{1+(1/m(|x|))-(1/2m(|x|))}{m(|x|)+2}$. Hence, nodes in the tree of all possible executions of $M^{\Pi}(x)$ that have two children correspond to a pair of prefixes $(y0, y1)$ that are each a prefix of a single solution output by $A(x)$ with high probability (i.e., probability exceeding $1/(m(|x|)+2)$), whereas these two different solutions must be in the set C_x (as per Definition 2). Hence, the number of different nodes in level i of the tree is upper-bounded by the number of different i-bit long prefixes of different solutions in C_x. It follows that the number of leaves in this trees is at most $|C_x| \leq m(|x|)$. ■

5 Acknowledgements and the story behind the dedication

I wish to thank Salil Vadhan and Ofer Grossman for their contributions to this work. In particular, the definition of m-pseudodeterministic list solver (i.e., Definition 5) was suggested to me by Salil Vadhan, and Part 2 of Theorem 8 was suggested to me by Ofer Grossman. I am grateful to both of them for the permissions to include their contributions here.

In the Fall of 2018, a group of former students of Shafi started planning to hold a small celebration of her birthday (November 14th). One day after November 14th 2018, I heard Shafi present an overview of pseudodeterminism, a direction of research she initiated in her paper with Gat [1]. It has occurred to me that nothing could be more appropriate than to try to contribute to this research direction. This work was first presented in the aforementioned celebration, which took place at the Simons Institute on January 13th 2019.

The dedication is meant to evoke the dedication of Beethoven's third ("Eroica") symphony, but avoiding Shafi's name is not meant to express dismay.

References

[1] E. Gat and S. Goldwasser. Probabilistic Search Algorithms with Unique Answers and Their Cryptographic Applications. In *ECCC*, TR11–136, 2011.

[2] O. Goldreich. *Computational Complexity: A Conceptual Perspective.* Cambridge University Press, 2008.

[3] O. Goldreich. In a World of P=BPP. In *Studies in Complexity and Cryptography*, LNCS Vol. 6650, Springer, pages 191–232, 2011.

[4] O. Goldreich, S. Goldwasser, and D. Ron. On the possibilities and limitations of pseudodeterministic algorithms. In *4th ITCS*, pages 127–138, 2013.

[5] S. Goldwasser and O. Grossman. Bipartite Perfect Matching in Pseudo-Deterministic NC. In *44th ICALP*, pages 87:1–87:13, 2017.

[6] O. Grossman and Y.P. Liu. Reproducibility and Pseudo-Determinism in Log-Space. In *30th SODA*, pages 606–620, 2019.

[7] M. Saks and S. Zhou. $\mathrm{BP_HSpace}(S) \subseteq \mathrm{DSPACE}(S^{3/2})$. *JCSS*, Vol. 58 (2), pages 376–403, 1999.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il