On the possibilities and limitations of pseudodeterministic algorithms

Oded Goldreich* Shafi Goldwasser† Dana Ron‡

November 1, 2012

Abstract

We study the possibilities and limitations of pseudodeterministic algorithms, a notion put forward by Gat and Goldwasser (2011). These are probabilistic algorithms that solve search problems such that on each input, with high probability, they output the same solution, which may be thought of as a canonical solution. We consider both the standard setting of (probabilistic) polynomial-time algorithms and the setting of (probabilistic) sublinear-time algorithms. Some of our results are outlined next.

In the standard setting, we show that pseudodeterminstic algorithms are more powerful than deterministic algorithms if and only if $\mathcal{P} \neq \mathcal{BPP}$, but are weaker than general probabilistic algorithms. In the sublinear-time setting, we show that if a search problem has a pseudodeterminstic algorithm of query complexity q, then this problem can be solved deterministically making $O(q^4)$ queries. This refers to total search problems. In contrast, for several natural promise search problems, we present pseudodeterministic algorithms that are much more efficient than their deterministic counterparts.

Keywords: Search problems, BPP, ZPP, unique solutions, sublinear-time computations

*Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel. oded.goldreich@weizmann.ac.il Partially supported by the Israel Science Foundation (grant No. 1041/08).

†MIT (Cambridge, MA, USA) and Weizmann Institute of Science (Rehovot, Israel). shafi.goldwasser@weizmann.ac.il

‡Department of EE–Systems, Tel-Aviv University, Ramat-Aviv, Israel. danar@eng.tau.ac.il Partially supported by the Israel Science Foundation (grant No. 246/08).

1 Introduction

In [5], Gat and Goldwasser initiated a study of probabilistic (polynomial-time) search algorithms that, with high probability, output the same solution. That is, for a fixed binary relation R, such algorithms associate a canonical solution, denoted $s(x)$, with any instance x such that, with overwhelmingly high probability, on input x the algorithm outputs $s(x)$.

Algorithms that satisfy the foregoing condition are called pseudodeterministic[1], because they essentially offer the same functionality as deterministic algorithms; that is, they produce a fixed (canonical) output for each possible input (except with negligible error probability). In contrast, arbitrary probabilistic algorithms that solve search problems may output different solutions when presented with the same input (but using different internal coin tosses); that is, on input x, with overwhelmingly high probability, the output is arbitrarily distributed among all valid solutions for x (e.g., it may be uniformly distributed).

Pseudodeterministic algorithms have the appealing feature that they (typically) yield the same result, when invoked on the same input. Thus if two parties invoke the algorithm on the same input at different times and locations (and using different sources of randomness), they are still guaranteed to obtain the same result, except very rarely. Needless to say, the lack of such agreement is a drawback of standard probabilistic algorithms.

The notion of a canonical solution is quite natural, and has appeared in various contexts such as the notion of canonical labeling of graphs, cf. [1]. Furthermore, probabilistic algorithms that find canonical solutions for natural problems (for which no deterministic algorithms is known) were implicit in work of Lenstra and de Smit [13].

In addition to defining pseudodeterministic algorithms and presenting pseudodeterministic polynomial-time algorithms for several natural search problems (which were previously known to be solvable probabilistically), Gat and Goldwasser [5] characterized the class of search problems having such algorithms. They showed that a search problem has a polynomial-time pseudodeterministic algorithm if and only if solving the search problem is reducible in deterministic polynomial-time to some set in $\mathcal{BPP}$ (see Theorem 2.2).

Many questions come to mind: Do polynomial-time pseudodeterministic algorithms exist for all search problems that can be solve in probabilistic polynomial-time? Or, to the other extreme, do polynomial-time pseudodeterministic algorithms exist only for search problems that can be solved in deterministic polynomial-time? And what happens in the context of sublinear time and query complexity?

The latter question is appealing since probabilism is essential to any sublinear-time algorithm that solves a "non-degenerate problem". Thus, unlike in the context of polynomial-time, there are a huge number of probabilistic sublinear-time algorithms (cf., the property testing literature). Furthermore, many property testing algorithms yield algorithms that solve related search problems (e.g., finding a k-partition with few violating edges [8]), and in all cases that come to mind these algorithms are not pseudodeterministic. An intriguing question is whether this is inherent.

1.1 Our results

In the current work, we advance the understanding of probabilistic algorithms for finding canonical solutions, considering both the standard context of polynomial-time algorithms and the context

[1] We mention that pseudodeterministic polynomial-time algorithms were also called Bellagio algorithms (in [5]).

of sublinear-time algorithms. In both cases, we explore the possibilities and limitations of such algorithms, but the results provided in the sublinear case are more natural.

The standard context (of PPT algorithms). The characterization of the class of search problems having pseudodeterministic algorithms (in terms of P-reducibility[2] to $\mathcal{BPP}$ (i.e., Theorem 2.2)) implies that pseudodeterminstic algorithms may exceed the scope of deterministic algorithms only if $\mathcal{BPP} \neq \mathcal{P}$. We first show that this necessary condition is also sufficient; in other words, if pseudodeterminstic algorithms do not exceed the scope of deterministic algorithms, then $\mathcal{BPP} = \mathcal{P}$. On the other hand, we show that there exist search problems that can be solved in probabilistic polynomial-time but have no pseudodeterminstic algorithms. Thus, in the context of solving search problems in polynomial-time, *pseudodeterminstic algorithms are more powerful than deterministic algorithms if and only if $\mathcal{P} \neq \mathcal{BPP}$, but are weaker than general probabilistic algorithms.*

We also discuss a zero-error version of the notion of pseudodeterministic algorithms; that is, these algorithms never output a solution that differs from the canonical one (but they may rather halt, with small probability, while providing a special error indication). We show that this class equals the class of search problems that are P-reducible to $\mathcal{ZPP}$.

The context of sublinear-time algorithms. We show several limitations on the scope of finding canonical solutions in probabilistic sublinear number of queries. Firstly, this is not possible for problems that have instances of linear sensitivity (i.e., search problems R having infinitely many instance x such that the set of solutions associated with x does not intersect the set of solutions associated with most strings obtained by flipping one bit of x). Note that many natural search problems have such instances. Secondly, with respect to non-adaptive algorithms, finding canonical solutions probabilistically is not easier than doing so deterministically. Lastly, even when adaptivity is allowed, finding canonical solutions probabilistically is not *significantly easier* than doing so deterministically. In particular, if this can be done in a constant or poly-logarithmic number of queries, then it can be done so deterministically. In general, the speed-up offered by probabilism (over determinism) is at most polynomial.

The aforementioned negative result implies a huge gap between probabilistic algorithms and pseudodeterministic ones: There exist search problems that are solved probabilistically in a constant number of queries but have no pseudodeterministic algorithm of sublinear query complexity. We also demonstrate the existence of a three-way separation between probabilism, pseudodeterminism, and determinism: There exist problems in which the query complexities are $O(1)$, n^c for some $c \in (0,1)$, and $\Omega(n)$, respectively.

The above negative results leave some room for sublinear-time pseudodeterministic algorithms. Firstly, pseudodeterministic algorithms may be somewhat faster than their deterministic counterparts, provided that they are adaptive and that the search problem has no instances of linear sensitivity. Secondly, the negative results refer to *total search problems*; that is, the search algorithm is required to work for all possible inputs (and not for a subset of them). In contrast, strong "disconnectivity" of the (promise) set of legitimate inputs allows probabilistic algorithms to find canonical solutions much more efficiently than deterministic ones. Several natural cases are presented in Section 4, and include search problems for graphs in the adjacency matrix model and in the incidence lists model. We discuss one, referring to the adjacency matrix model, next.

[2] A P-reduction is a deterministic polynomial-time Turing reduction.

Suppose that we are presented with a huge graph G that is obtained by a blow-up of some small unknown graph H (i.e., each vertex of H is replaced by a cluster of $m \in [M, 2M]$ vertices and edges of H are replaced by complete bipartite graphs). That is, G has some underlying structure (captured by H), and we are interested in finding this underlying structure (or finding substructures in it). Then, a pseudodeterministic algorithm may have complexity related to the size of H, whereas the complexity of any deterministic algorithm will be related to the size of G.

1.2 Organization

In Section 2 we present the basic definitions as well as the characterization result of [5], which are the starting point of our study. In Section 3, we consider the standard context of polynomial-time algorithms, whereas in Section 4 we consider the context of sublinear time and query algorithms. Finally, in Section 5, we discuss a zero-error version of the notion of pseudodeterministic algorithms.

2 Background

Our starting point is the following notion, explicitly introduced by Gat and Goldwasser [5]. As usual, the definition refers to a fixed error probability (of 1/3), and the ramification for other error bounds is straightforward.

Definition 2.1 (finding canonical solutions and pseudodeterministic algorithms [5]):[3] *Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation, and let $R(x) = \{y : (x,y) \in R\}$ and $S_R = \{x : R(x) \neq \emptyset\}$. Suppose that the empty string is never a solution; that is, $\lambda \notin R(x)$ for all x. A* (possibly probabilistic) *algorithm A* finds canonical solutions with respect to *R if there exists a function $s : \{0,1\}^* \to \{0,1\}^*$ such that the following conditions hold:*

1. *For every $x \in S_R$ it holds that $s(x) \in R(x)$, whereas $s(x) = \lambda$ for every $x \in \{0,1\}^* \setminus S_R$.*
2. *For every $x \in \{0,1\}^*$, it holds that $\Pr[A(x) = s(x)] \geq 2/3$.*

The function s is called a canonical solution function for *R and A is called a* pseudodeterministic algorithm for *R.*

In the context of polynomial-time algorithms, Definition 2.1 is robust in the sense that replacing the threshold 2/3 by either $0.5 + 1/\mathrm{poly}(|x|)$ or $1 - \exp(-\mathrm{poly}(|x|))$ yields the same class (via standard error reduction). In other settings (where the complexity measures are more refined), the same error reduction applies, but one should bear in mind its cost (e.g., going from error probability of 1/3 to error probability δ costs a multiplicative factor of $O(\log(1/\delta))$ in all complexities). Also, any deterministic algorithm for finding solutions w.r.t R finds canonical solutions w.r.t R.

Of course, a probabilistic (polynomial-time) algorithm for finding canonical solutions w.r.t R is of interest only if it is (significantly) faster than the known deterministic algorithm (e.g., if no deterministic (polynomial-time) algorithms are known for finding solutions w.r.t R). Gat and Goldwasser presented several such examples as well as established the following necessary and sufficient condition for such algorithms:

[3] An alternative formulation may refer to R' such that for every x it holds that $R'(x) \neq \emptyset$ and either $\lambda \notin R'(x)$ or $R'(x) = \{\lambda\}$. Indeed, $R'(x) = R(x)$ if $x \in S_R$ and $R'(x) = \{\lambda\}$ otherwise.

Theorem 2.2 (a characterization [5]): *A relation R has a pseudodeterministic polynomial-time algorithm if and only if finding solutions w.r.t R is P-reducible to some set in $\mathcal{BPP}$, where a* P-reduction *is a deterministic polynomial-time Turing reduction.*

In particular, if $\mathcal{BPP} = \mathcal{P}$, then the existence of a probabilistic polynomial-time algorithm for finding canonical solutions w.r.t R implies that (canonical) solutions w.r.t R can be found by a deterministic polynomial-time algorithm.

Proof: The first direction is proved by using the P-reduction as an algorithm, while providing answers to its queries via a BPP algorithm (of sufficiently low error probability). This yields a probabilistic algorithm that outputs the same (canonical) solution whenever all the emulated answers are correct. For the opposite direction, suppose that algorithm A finds canonical solutions w.r.t R, and consider the set S such that $(x, i, b) \in S$ if and only if $\Pr[A(x)_i = b] \geq 2/3$, where $A(x)_i$ denotes the i^{th} bit in the output $A(x)$. Then, (i) the set S is a BPP set (since it is essentially decided by $A(x)$), and (ii) finding solutions w.r.t R is P-reducible to S. ■

Note: A zero-error version of Definition 2.1 and a corresponding chractaterization (in terms of $\mathcal{ZPP}$) are discussed in Section 5.

3 The standard context (of PPT algorithms)

In this section we consider the standard setting of polynomial-time algorithms.

3.1 Possibilities and impossibilities

Theorem 2.2 implies that relations for which canonical solutions can be found in probabilistic polynomial-time but not in deterministic polynomial-time may exist only if $\mathcal{BPP} \neq \mathcal{P}$. We observe that this necessary condition is also sufficient.

Proposition 3.1 (on the non-triviality of pseudodeterministic algorithms): *If $\mathcal{BPP} \neq \mathcal{P}$, then there exists a search problem that has a pseudodeterministic polynomial-time algorithm but does not have a deterministic polynomial-time algorithm. Furthermore, for every $S \in \mathcal{BPP}$, there exists a relation R such that (1) deciding S is P-reducible to finding solutions w.r.t R, and (2) canonical solutions w.r.t R can be found in probabilistic polynomial-time.*

Proof: The main claim follows from the furthermore claim. To see this consider, for every $S \in \mathcal{BPP}$, the relation R that is guaranteed by the furthermore claim, and suppose (towards the contradiction) that whenever canonical solutions can be found in probabilistic polynomial-time they can also be found in deterministic polynomial-time. Then, S is in $\mathcal{P}$ (by combining the guaranteed P-reduction of S to R with the deterministic algorithm for finding solutions w.r.t R).

We now turn to the furthermore claim itself. This claim is proved by observing that any decision problem in $\mathcal{BPP}$ is essentially a search problem that has a pseudodeterministic polynomial-time algorithm. Specifically, for any $S \in \mathcal{BPP}$, consider the relation

$$R = \{(x, 1) : x \in S\} \cup \{(x, 0) : x \in \{0,1\}^* \setminus S\} \tag{1}$$

Clearly, S is P-reducible to solving R, which has unique solutions, which in turn can be found probabilistically (by running the decision procedure for S). Needless to say, various syntactic requirements regarding R can be met by simple modifications; for example, if solutions have to be longer than the input and non-unique, then one may use $R'(x) = \{\sigma^{|x|}\alpha : \sigma \in R(x) \ \& \ \alpha \in \{0,1\}^{|x|}\}$. ■

An impossibility result. On the other hand, there exist search problems that can be solved in probabilistic polynomial-time but for which canonical solutions cannot be found in probabilistic polynomial-time.

Proposition 3.2 (search problems solvable in PPT, but not pseudodeterministically): *There exists a search problem R such that R can be solved in probabilistic polynomial-time but no probabilistic algorithm can find canonical solutions w.r.t R.*

Proof: Let $K(z)$ denote the Kolmogorov complexity of z, and consider the binary relation

$$R = \{(x,y) : |y| = 2|x| \ \& \ K(y) > 2|x| - 7\}. \tag{2}$$

(Indeed, the solutions to x only depend on the length of x.) Then, on the one hand, solutions w.r.t R can be found in probabilistic polynomial-time (by just generating uniformly distributed $2|x|$-bit long strings, and hitting a valid solution with probability at least 99%). But, on the other hand, no probabilistic algorithm can find canonical solutions w.r.t R, because a probabilistic t-time algorithm A that finds a canonical solution y with probability greater than half yields a deterministic (indeed, $\exp(t)$-time) algorithm that (on input x) finds this y (which implies $K(y) \leq O(1) + |x|$, in contradiction to $K(y) > 2|x| - 7$). ■

On problems with unique solution. Clearly, for any relation R having unique solutions (i.e., $|R(x)| \leq 1$ for every x), the unique solutions are canonical (i.e., whenever $R(x) = \{y\}$ we must use $s(x) = y$). Thus, a probabilistic algorithm that finds solutions w.r.t R, actually finds canonical solutions w.r.t R. Many such algorithms are known, especially in the domain of computational number theory (e.g., finding (the smallest) square root modulo a given prime, or the subexponential-time algorithms for factoring and discrete logarithms). Hence, while we are still interested in new probabilistic algorithms that solve natural search problems with unique solutions (and, in general, in any new probabilistic algorithms that solve natural computational problems), such algorithms are not necessarily "good" examples for the concept of finding canonical solutions.

3.2 On BPP-search problems

A natural class of "BPP search problems" (studied in [7]) refers to the class of all search problems that can be solved in probabilistic polynomial-time and for which solutions can be recognized in probabilistic polynomial-time.

Definition 3.3 (BPP-search problems): *A relation R is a* BPP-search problem *if (1) the set R is in $\mathcal{BPP}$, and (2) there exists a probabilistic polynomial-time algorithm A such that for every $x \in S_R$ it holds that* $\Pr[A(x) \in R(x)] \geq 2/3$.

(There is no need to require that for every $x \notin S_R$ it holds that $\Pr[A(x) = \lambda] \geq 2/3$, since (by using the algorithm of Item (1)) we can guarantee that wrong solutions are output with probability at most 1/3.) Note that the relation used in Proposition 3.2 is not a BPP-search problem (since it does not satisfy the first condition (i.e., it is not recognizable in probabilistic polynomial-time)). A natural question is whether every BPP-search problem has a polynomial-time pseudodeterministic algorithm. We note that a positive answer would follow if the "promise problem version of BPP", denoted pr$\mathcal{BPP}$, were P-reducible to $\mathcal{BPP}$ itself. This is the case since Goldreich [7] showed that BPP-search problems are P-reducible to pr$\mathcal{BPP}$, whereas by Theorem 2.2 every search problem that is P-reducible to $\mathcal{BPP}$ has a a polynomial-time pseudodeterministic algorithm.[4] On the other hand, a negative answer must assume that pr$\mathcal{BPP}$ is not solvable in deterministic polynomial-time.

Not being able to resolve the above question, we present a search problem that is "semi-complete" with respect to the class of BPP-search problems in the sense that (i) if this problem has a polynomial-time pseudodeterministic algorithm, then so does any problem in the class, and (ii) the problem itself is "somewhat related" to this class.[5]

The problem we present is "generic" and is thus denoted G: For a Boolean circuit C, let $^{p}_{C}(y) \stackrel{\text{def}}{=} \Pr_z[C(yz) = 1]$, where $|z| = |y|$ (i.e., we assume that C takes an even number of input bits, and denote a generic prefix by y and suffix by z). Then, $G(C) \stackrel{\text{def}}{=} \{y : {}^{p}_{C}(y) \geq 2/3\}$ if $\Pr_r[{}^{p}_{C}(r) \geq 2/3] \geq 2/3$, and $G(C) \stackrel{\text{def}}{=} \emptyset$ otherwise. This search problem can be solved by merely selecting a random r, since $\Pr_r[{}^{p}_{C}(r) \geq 2/3] \geq 2/3$ for any $C \in S_C$. Thus, G satisfies the second condition of Definition 3.3. It is unclear whether G satisfies the first condition, but it does satisfy a weak version of this condition: One can distinguish in probabilistic polynomial-time between pairs (C, y) in G and pairs (C, y) such that either ${}^{p}_{C}(y) < 0.66$ or $\Pr_r[{}^{p}_{C}(r) > 0.66] < 0.66$.

Proposition 3.4 *Every BPP-search problem is P-reducible to G.*

Thus, if G has a polynomial-time pseudodeterministic algorithm, then we obtain such an algorithm for any BPP-search problem.

Proof: For any BPP-search problem R, we show a P-reduction of R to G. Fixing R, let V and F be the probabilistic polynomial-time algorithms guaranteed by the two conditions of Definition 3.3 (i.e., V recognizes valid instance solution pairs, whereas F finds valid solutions). Assume, without loss of generality, that each of these algorithms uses the same number of coins (on the respective inputs). Then, given an input x, consider the circuit C_x that represents the computation of $V(x, F(x))$; that is, $C_x(r, r')$ is the output of V on input (x, s) and coins r', where s is the output of F on input x and coins r. (By the above, $|r'| = |r|$.) Note that ${}^{p}_{C_x}(r) = \Pr_{r'}[C_x(r, r) = 1] = \Pr[V(x, F(x; r)) = 1]$, where $F(x; r)$ denotes the output of F on input x and coins r.

Note that if $x \in S_R$, then $\Pr_r[{}^{p}_{C_x}(r) \geq 2/3] \geq 2/3$, which implies that $G(C_x) \neq \emptyset$. On the other hand, $G(C_x) \subseteq \{r : F(x; r) \in R(x)\}$, since $r \in G(C_x)$ implies ${}^{p}_{C_x}(r) \geq 2/3$. Hence, a P-reduction of R to G may proceed as follows. On input x, it constructs the circuit C_x, invokes the G-solver to obtain a solution, denoted r, for C_x w.r.t G, and outputs $F(x; r)$. ■

[4] We note that, in contrast to Theorem 2.2, a P-reduction of BPP-search problems to pr$\mathcal{BPP}$ (and, in particular, the one in [7]) does not seem to yield an algorithm for finding canonical solutions: The problem is that queries that violate the promise may yield arbitrary answers, which may possibly lead the search to different solutions.

[5] Indeed, it would have been better to present a problem in BPP-search that is complete under P-reductions, but this seems as difficult as presenting a problem in $\mathcal{BPP}$ that is complete under P-reductions.

4 The context of sublinear-time algorithms

Let us now turn to the sublinear-time model. In this context, the algorithm is given oracle access to the input, and is only provided with the input's length as explicit input. Thus, the computation of machine M with oracle access to x is captured by the notation $M^x(|x|)$.

4.1 Limitations

We show several limitations on scope of finding canonical solutions in probabilistic sublinear-time. Firstly, this is not possible for problems that have instances of linear sensitivity (i.e., search problems R having infinitely many $x \in S_R$ such that $|\{i \in [|x|] : R(x) \cap R(x \oplus e_i) = \emptyset\}| = \Omega(|x|)$, where $e_i = 0^{i-1}10^{|x|-i}$). Note that many natural search problems have such instances (e.g., the number of connected components in a graph may change if a single edge is added in any of many possible ways). Secondly, finding canonical solutions probabilistically is not significantly easier than doing so deterministically. In particular, if this can be done in a constant or poly-logarithmic number of queries, then it can be done so deterministically.

Theorem 4.1 (on the limitations of sublinear-time algorithms): *Let R be an arbitrary search problem such that $S_R = \{0,1\}^*$ and M be a probabilistic machine that makes at most $q(n)$ queries on any n-bit long input.*

1. *If for every function $f : \{0,1\}^* \to \{0,1\}^*$ there exists an x such that either $f(x) \notin R(x)$ or $|\{i \in [|x|] : f(x) \neq f(x \oplus e_i)\}| > 3q(|x|)$, then M fails to find canonical solutions w.r.t R. In particular, if there exists an x such that $|\{i \in [|x|] : R(x) \cap R(x \oplus e_i) = \emptyset\}| > 3q(|x|)$, then M fails to find canonical solutions w.r.t R.*

2. *If M is non-adaptive and finds canonical solutions w.r.t R, then canonical solutions w.r.t R can be found deterministically by making at most $3q(n)$ non-adaptive queries.*

3. *If M finds canonical solutions w.r.t R, then canonical solutions w.r.t R can be found deterministically by making at most $\min(28q(n)^4, 27\ell(n)q(n)^3)$ queries* (possiblly adaptively)*, where $\ell(n) = \lceil \log_2 |\cup_{x \in \{0,1\}^n} R(x)| \rceil$.*

The condition $S_R = \{0,1\}^*$ can be assumed, without loss of generality, since, for any R', we may consider R such that $R(x) = \{1y : y \in R'(x)\}$ if $x \in S_{R'}$ and $R(x) = \{0^{|x|}\}$ otherwise (cf. Footnote 3). The polynomial blow-up in Item 3 is inherent, since the deterministic query complexity is greater than $\Omega(q(n)^{1.3})$ even in the special case of decision problems (which correspond to search problems with unique solutions in $\{0,1\}$). This follows from a gap between the probabilistic and deterministic decision tree complexity [16] (see also [15]). Item 3 is established by using an upper bound on the very same question, which asserts that the deterministic decision tree complexity is at most cubic in the probabilistic complexity [14].

Proof: Consider an arbitrary probabilistic machine M, and let $Q(x)$ denote a random variable that represents the set of queries made by $M^x(|x|)$; that is, for every coin sequence ω, let $Q_\omega(x)$ denote the set of queries made by $M^x(|x|, \omega)$, and let $Q(x) \leftarrow Q_\omega(x)$ for a uniformly selected ω. Then, $H(x) \stackrel{\text{def}}{=} \{i \in [|x|] : \Pr[i \in Q(x)] \geq 1/3\}$ has cardinality at most $3q(n)$. The basic observation is that if $\Pr[i \in Q(x)] < 1/3$, then, for every y, it holds that $|\Pr[M^x(|x|) = y] - \Pr[M^{x \oplus e_i}(|x|) = y]| < 1/3$.

The first item is proved by considering all possible functions f that may serve as a canonical solution function w.r.t R; that is, $f(x) \in R(x)$ for all x's. For each such function f, we are guaranteed (by the hypothesis) that there exists an x such that $|\{i \in [|x|] : f(x) \neq f(x \oplus e_i)\}| > 3q(|x|)$. Fixing such an x, it follows that there exists an $i \in [|x|]$ such that $f(x) \neq f(x \oplus e_i)$ and $\Pr[i \in Q(x)] < 1/3$ (since there are at least $3q(n)+1$ indices that satisfy the first condition and at most $3q(n)$ violate the second). In such a case, it holds that $\Pr[M^{x \oplus e_i}(|x|) = f(x)] > \Pr[M^x(|x|) = f(x)] - 1/3$, which implies that $\Pr[M^{x \oplus e_i}(|x|) = f(x \oplus e_i)] < (4/3) - \Pr[M^x(|x|) = f(x)]$. It follows that f cannot be the canonical solution function of M, since either $\Pr[M^x(|x|) = f(x)] < 2/3$ or $\Pr[M^{x \oplus e_i}(|x|) = f(x \oplus e_i)] < 2/3$. Given that the same argument holds with respect to any possible canonical solution function (w.r.t R), it follows that M has no such function (which means that it is not a pseudodeterministic algorithm for R).

Turning to the second item, we observe that, in the non-adaptive case, $H(x)$ depends only on the length of x, and so we may define $H_n \stackrel{\text{def}}{=} H(1^n)$ and note that $H(x) = H_n$ for all $x \in \{0,1\}^n$. We claim that every $x, x' \in \{0,1\}^n$ that agree on bit locations H_n must admit the same canonical solution. Once this claim is proven, the second item follows, because a non-adaptive deterministic machine may just query the oracle on positions in H_n, set the other input bits arbitrarily, and decide accordingly (by emulating all possible executions of the probabilistic algorithm internally).

The foregoing claim (i.e., inputs that agree on bits in H_n must admit the same canonical solution) is proved by considering a "Hamming walk" from x to x' (i.e., considering a sequence $x = v_1, v_2, \ldots, v_t = x'$ such that each pair (v_i, v_{i+1}) differ in a single bit), and using $\Pr[M^{v_{i+1}}(n) = y] > \Pr[M^{v_i}(n) = y] - 1/3$. Specifically, suppose that y is a canonical solution found by M^x (i.e., $\Pr[M^x(n) = y] \geq 2/3$), then y must also be a canonical solution for v_2 (since $\Pr[M^{v_2}(n) = y] > 1/3$), and the same holds for $v_3, \ldots, v_t$ (since if y is a canonical solution for v_i, then $\Pr[M^{v_i}(n) = y] \geq 2/3$, which implies $\Pr[M^{v_{i+1}}(n) = y] > 1/3$, which in turn implies that y is a canonical solution for v_{i+1}).

Turning to the third item, we define $S_n = \{s(x) : x \in \{0,1\}^n\}$, where $s : \{0,1\}^* \to \{0,1\}^*$ describes the canonical solutions used by M. We shall prove (below) that $|S_n| < 2^{q(n)+1}$. Now, consider an arbitrary (deterministic) *binary decision tree T_n of depth $q(n)+1$ for the set S_n*, where each internal node v of the tree is associated with some subset $S_v \subseteq S_n$ as well as a partition of this subset into two equals subsets, denoted (S_v^0, S_v^1) associated with its children (i.e., if the children are v_0 and v_1, then $S_{v_\sigma} = S_v^\sigma$ for each $\sigma \in \{0,1\}$). Machine M can be used to decide each of the corresponding "bits" (since it actually yields the entire value of $s(\cdot)$). Formally, for every internal node v of T_n, we consider an oracle machine M_v, derived from M, that probabilistically decides the corresponding bit (i.e., $M_v^x(n) = \sigma$ if and only if $s(x) \in S_v^\sigma$) by making at most $q(n)$ queries (to x).[6] Replacing M_v by a deterministic counterpart (denoted M'_v) that is guaranteed by Nisan's result [14], we obtain a deterministic algorithm that (on input x) finds $s(x)$ by making at most $(q(n)+1) \cdot 27q(n)^3$ queries. (That is, our algorithm goes down T_n by using the various machines M'_v to make the relevant decisions.)

It is thus left to show that $|S_n| < 2^{q(n)+1}$. For each $y \in S_n$, let $\chi_y(\omega) = 1$ if there exists an input $x \in \{0,1\}^n$ such that machine M outputs y on input x when using coins ω, and $\chi_y(\omega) = 0$ otherwise. Clearly, $\sum_y \chi_y(\omega) \leq 2^{q(n)}$, for any ω, and so $\sum_y \Pr[\chi_y(\omega) = 1] \leq 2^{q(n)}$. Assuming towards the contradiction that $|S_n| \geq 2^{q(n)+1}$, we obtain an $x \in \{0,1\}^n$ such that $\Pr[\chi_{s(x)}(\omega) = 1] \leq 1/2$, which implies $\Pr[M^x(n) = s(x)] \leq 1/2$ (in violation of the correctness of M). The theorem follows. ■

[6] In the next line, we consider M_v as a depth $q(n)$ probabilistic decision tree. This tree should not be confused with the tree T_n introduced above.

Corollary 4.2 (Corollary to Theorem 4.1): *There exist search problems that can be solved probabilistically in a constant number of queries, but have no pseudodeterministic algorithm of sublinear query complexity.*

Proof: Consider, for example, the search problem $R \stackrel{\text{def}}{=} \{(x,v) : |\mathtt{wt}(x)-v| < |x|/10\}$, where $\mathtt{wt}(x)$ is the Hamming weight of x. Clearly, R can be solved probabilistically with a constant number of queries, and we shall use Item 1 of Theorem 4.1 to show that it has no pseudodeterministic algorithm that makes less than $|x|/4$ queries. Towards this end, we have to show that the hypothesis of Item 1 (of Theorem 4.1) holds for $q(n) = n/4$. Indeed, let f be a function such that $f(x) \in R(x)$ holds for every x. Starting at $x_0 = 0^n$, we seek an x such that $|\{i \in [|x|] : f(x) \neq f(x \oplus e_i)\}| > 3n/4$. We call such an x unstable. We proceed in $n/4$ iterations such that (in the j^{th} iteration) if the current x_{j-1} is not unstable, then we select x_j in $N_j \stackrel{\text{def}}{=} \{x_{j-1} \oplus e_i : i \in [n] \ \&\ \mathtt{wt}(x_{j-1} \oplus e_i) = \mathtt{wt}(x_{j-1})+1\}$ such that $f(x_j) = f(x_{j-1})$, which is possible since the set N_j has cardinality $n - j + 1 > 3n/4$. As long as the process goes on, we have $f(x_j) = f(x_0)$, which implies that we must find a string that is unstable (since $R(x_0) \subseteq [0, 0.1n]$, whereas $R(x_{n/4}) \subseteq [0.15n, 0.35n]$). ■

Proposition 4.3 (probabilism vs pseudodeterminism vs determinism): *There exists a search problem R such that*

1. *R can be solved probabilistically in a constant number of queries;*
2. *R has a pseudodeterministic algorithm of query complexity $n^{0.76}$, but no pseudodeterministic algorithm of query complexity $n^{0.33}$;*
3. *R cannot be solved deterministically in sublinear query complexity.*

Proof: By [16, 15], there exists a Boolean function, denoted b, that cannot be computed deterministically in a sublinear number of queries but can be computed probabilistically (with error probability at most 1/3) in $n^{0.76}$ queries; Let $R = R_1 \cup R_2$, where $R_1 = \{(x, b(x)) : x \in \{0,1\}^*\}$ and $R_2 = \{(x,i) \in \{0,1\}^* \times \mathbb{N} : |i - 10\rho(x)| \leq 1\}$, where $\rho(x) = \mathtt{wt}(x)/|x|$. (Assume that $v \in R_2(x)$ is encoded by several bits and so is different from the single bit solution to R_1.) Then, R can be solved probabilistically in a constant number of queries (via R_2) and has a pseudodeterministic algorithm of query complexity $n^{0.76}$ (via R_1). To see that R cannot be solved deterministically in a sublinear number of queries, note that we may "force" the solver to solve R_1. Specifically, consider a (sublinear-query) deterministic algorithm that on some input x outputs a solution y. Then, y cannot be in $R_2(x)$, because this algorithm also outputs y on any string x' that agrees with x on the $o(n)$ queries bits (whereas $R_2(x) \cap R_2(x) = \emptyset$ for some of these x'). Lastly, the lower bound on the query complexity of pseudodeterministic algorithms for R follows by combining the deterministic lower bound with Item 3 of Theorem 4.1, while using $|\cup_{x \in \{0,1\}^n} R(x)| < 15$. ■

Discussion. Theorem 4.1 leaves some room for sublinear-time probabilistic algorithms for finding canonical solutions. Such algorithms may be somewhat faster than their deterministic counterparts, provided that they are adaptive and that the search problem has no instances of linear sensitivity. Even in these cases, the speed-up offered by probabilism (over determinism) is at most polynomial, but on the other hand polynomial differences do matter (also in the sublinear-time context, cf., e.g., [11]).

Theorem 4.1 refers to *total search problems*; that is, the searching algorithm is required to work for all possible inputs (and not for a subset of them). The claim of Item 2 holds whenever the set of legitimate inputs is sufficiently connected (w.r.t the Boolean lattice).[7] In contrast, disconnectivity of this (promise) set allow probabilistic algorithms to find canonical solutions more efficiently than deterministic ones.

The role of a promise can be demonstrated by considering the problem of estimating the fraction of 1-values in a binary string. As shown in the proof of Corollary 4.2, no sublinear-time probabilistic algorithm can find a canonical solution w.r.t the relation $R \stackrel{\text{def}}{=} \{(x,v) : |\mathtt{wt}(x) - v| < |x|/10\}$. However, if we restrict the set of admissible x's to those that satisfy $\mathtt{wt}(x) \in \{i \cdot (|x|/10) \pm s : i \in [10], s \in [|x|/30]\}$, then canonical solutions w.r.t the resulting promise problem can be found by a constant-time probabilistic algorithm. More interesting examples are presented next.

4.2 Possibilities

In light of Theorem 4.1, we focus on search problems with a promise. The promise consists of several "instance regions" such that only (distant) regions can be associated with different canonical solutions. That is, the allowed instances are partitioned into sets $I_1, \ldots, I_M \subset \{0,1\}^n$ such that the canonical solution for each I_j is the same and strings in different I_j's are far apart.

The above formulation is closely related to error correcting codes, and it suggests that we may get search problems for which canonical solutions can be found in sublinear-time by considering locally decodable codes. For details see the appendix. These examples refer to somewhat artificial objects (i.e., (corrupted) codewords w.r.t some error correcting codes). As in the context of property testing, the real question is whether we can present examples that refer to natural objects, under a natural representation. It seems that natural examples arise in the case of huge structures that are based on some small substructure. We consider such cases in two natural models of graphs.[8]

4.2.1 Graph problems in the adjacency matrix representation

A very natural model of direct access to graphs refers to providing oracle access to their adjacency predicate; that is, a query of the form (u,v) is answered by 1 if there exists an edge between vertices u and v in the graph, and by 0 otherwise. In the context of sublinear-time algorithms, this model is most suitable for dense graphs (cf. [8]).

In this model, a natural notion of a small substructure is provided by the notion of graph blow-up. Indeed, the notion of a graph blow-up yields natural examples of problems for which canonical solutions can be found in probabilistic constant time (but not in deterministic sublinear-time).

Graph blow-up. Recall that a blow-up of a graph $H = ([n], L)$ is a graph $G = ([N], E)$ obtained by replacing each vertex $i \in [n]$ of H by a (non-empty) subset $C_i \subset [N]$ (which may be called a cloud or a cluster) such that $(C_1, \ldots, C_n)$ is a partition of $[N]$, and placing a complete bipartite

[7] The claim of Item 1 holds when modified to the promise set (i.e., x and the $x \oplus e_i$'s have to satisfy the promise). Regarding Item 2, consider, for example, the promise problem that consists of finding a 1-entry in a string x that contains a majority of 1-entries. Clearly, this promise problem can be solved probabilistically with a constant number of queries, but cannot be solved deterministically in less than $|x|/2$ queries. By an extension of Item 2 of Theorem 4.1, this problem cannot be solved by a non-adaptive pseudodeterministic algorithm that makes less than $|x|/6$ queries. The claim extends to adaptive algorithms by observing that any adaptive pseudodeterministic algorithm for this problem can be converted into a non-adaptive one, while maintaining the query complexity (cf. [2]).

[8] Interestingly, similar substructures were used in the (different) context of [9].

graph between C_i and C_j if and only if $(i,j) \in L$. That is, if $(i,j) \notin L$, then there will be no edges between C_i and C_j (and, in particular, there are no edges within each C_i). One usually looks at balanced blow-ups, in which all clouds are of the same size, but we may look at ϵ-balanced blow-up in which each cloud contains at least an ϵ fraction of the vertices (i.e., $|C_i| \geq \epsilon N$ for each $i \in [n]$). Finally, call H irreducible if it is not a blow-up of any smaller graph.

Now, consider an arbitrary (solvable) search problem R for graphs (e.g., finding spanning forests, shortest paths trees, Eulerian or Hamiltonian cycles, maximum matching, minimum vertex cover, etc). Define a parameterized search problem $\mathcal{BL}_{k,\epsilon}(R)$ as follows:

Input: A graph $G = ([N], E)$ that is an ϵ-balanced blow-up of some irreducible k-vertex graph.

Valid solution: Any solution y for a graph H (i.e., $(H,y) \in R$), where H is a k-vertex graph such that G is an ϵ-balanced blow-up of H.

(Indeed, this is a promise problem, and it is parameterized by k and ϵ.) Note that ϵ-balanced blow-ups provide a reasonable model for some types of clustering situations; the search problems R relate to the structure that underlies these clusters.

Proposition 4.4 (pseudodeterministic algorithms for $\mathcal{BL}_{k,\epsilon}(R)$): *For every R, k, and $\epsilon > 0$ as above, canonical solutions w.r.t $\mathcal{BL}_{k,\epsilon}(R)$ can be found by a probabilistic algorithm that makes $O(\epsilon^{-1} \log k)^2$ adjacency queries.*

Typically (i.e., ignoring pathological cases such as $\epsilon \geq 1/k$ or a trivial R), no deterministic algorithm making $N/4$ queries can find solutions w.r.t $\mathcal{BL}_{k,\epsilon}(R)$, since such an algorithm may never encounter any vertex of the smallest cloud.

Proof: Consider a randomized algorithm that uniformly selects a set S of $O(\epsilon^{-1} \log k)$ vertices in G, queries all vertex pairs, finds the unlabeled irreducible graph H such that the induced subgraph of G (i.e., G_S) is a blow-up of H, labels H in a canonical way, and finds a solution by running a deterministic algorithm on H. Note that the graph H that emerges from the sample is unlabeled and it is crucial for our application to obtain a canonical labeling of it (since the labeling will effect the solution found for H). However, a canonical labeling of a k-vertex graph can always be found deterministically in time $k!$ (e.g., by finding the lexicographically first representation among all possible labeling of H).

The analysis amounts to observing that, with high probability, the initial sample S will hit each cloud of the graph G, since G is an ϵ-balanced blow-up of some (irreducible) k-vertex graph. The irreducibility condition guarantees that, in that case, the graph H is uniquely determined by G_S (i.e., the subgraph of G induced by S). ■

Note. It is possible to relax the formalism in various ways, while maintaining the result (i.e., a pseudodeterministic algorithm of poly(k/ϵ) query complexity). For example, we may allow the input to be any graph that can be obtained from an ϵ-balanced blow-up of some irreducible k-vertex graph by changing at most $\epsilon N/6$ of the neighbors of each vertex.[9]

[9] In such a case, vertices that belong to the same cloud will differ on at most $\epsilon N/3$ neighbors, while vertices that belong to different clouds will differ on at least $2\epsilon N/3$ neighbors. Note that if we can change $\epsilon N/2$ neighbors of $2\epsilon N$ vertices, then the graph H may no longer uniquely recovered from G.

4.2.2 Graph problems in the incidence-lists representation

A standard model of direct access to graphs refers to providing oracle access to their incidence function; that is, a query of the form (v, i) is answered by the i^{th} neighbor of v if v has at least i neighbors, and by 0 otherwise. In the context of sublinear-time algorithms, this model is most suitable for bounded-degree graphs (cf. [10]).

In this model, a natural notion of a small substructure is provided by the notion of recurring connected components; for example, we may consider graphs that consist of connected components of bounded size, and further restrict the graphs such that each subgraph either appears in very few connected component or appears in many such components. Indeed, this notion yields natural examples of problems for which canonical solutions can be found in probabilistic constant time (but not in deterministic sublinear-time).

Specifically, consider an arbitrary search problem R for graphs, and define a parameterized search problem $\mathcal{RP}_{k,\epsilon}(R)$ as follows:

Input: A graph $G = ([N], E)$ that consists of connected components such that (1) each connected component is of size at most k, and (2) each connected component is isomorphic to a a number of connected components that is either greater than ϵN or smaller than $\epsilon N/2$.

Valid solution: Any solution y for a graph H (i.e., $(H, y) \in R$), where H is isomorphic to at least ϵN connected components of G. (If there is no such H, then the solution is a special symbol.)

Proposition 4.5 (pseudodeterministic algorithms for $\mathcal{RP}_{k,\epsilon}(R)$): *For every R, k, and $\epsilon > 0$ as above, canonical solutions w.r.t $\mathcal{RP}_{k,\epsilon}(R)$ can be found by a probabilistic a algorithm that makes $\widetilde{O}(k^2/\epsilon)$ incidence queries.*

Typically (i.e., ignoring pathological cases such as $\epsilon \geq 1/k$ or a trivial R), no deterministic algorithm making $\epsilon N/4$ queries can find solutions w.r.t $\mathcal{BL}_{k,\epsilon}(R)$, since such an algorithm cannot distinguish subgraphs that are isomorphic to at least ϵN connected components from subgraphs that are isomorphic to less than $\epsilon N/2$ such components.

Proof: Consider a randomized algorithm that samples $\widetilde{O}(\epsilon^{-1} \log k)$ vertices in G, and explores their connected component (by making at most k^2 incidence queries). The algorithm also estimates the number of connected components (in G) that are isomorphic to each k'-vertex possible graph, for k', such that all estimates are correct in the sense discussed below. Next, among the graphs that had an estimate above $3\epsilon N/4$, the algorithm selects one graph (in a canonical fashion), and labels it (canonically, as in the proof of Proposition 4.4). Finally, it finds the desired canonical solution by running a deterministic algorithm on this graph.

The analysis amounts to observing that, with high probability, the initial sample will provide adequately good estimates of the various subgraphs that appear as connected components. Specifically, note that by taking $O(\epsilon^{-1} \log(1/\delta))$ one can estimate with probability at least $1 - \delta$ the frequency of events such that events that have frequency below $\epsilon/2$ (resp., at least ϵ) are judged to have frequency below $3\epsilon/4$ (resp., above $3\epsilon/4$). ■

Note. It is possible to generalize the formalism and extend the results in various ways. For example, we may consider search problems R that refer to sets of graphs (e.g., we may seek a sequence of solutions such that each solution correspond to one of the graphs). Alternatively, we

may consider highly connected components that are connected via a structure of lower connectivity (e.g., the superstructure may be a ring or a two-dimensional grid and its nodes may be 5-connected graphs).

4.3 On the class of graph problems having pseudodeterministic algorithms

The examples presented in Section 4.2 correspond to search problems that refer to small substructures of graphs. We claim that this is no coincidence in the sense that pseudodeterministic algorithms of low query complexity may exist only for problems that are defined in terms of the frequency in which various small graphs appear as induced subgraphs. Our discussion is restricted to "search problems regarding graphs" and to algorithms that access graphs via their adjacency predicate (i.e., as in §4.2.1).

A natural notion of "search problems regarding graphs" corresponds to binary relations R such that if $y \in R(G)$ then y is a solution for any graph that is isomorphic to G. For example, $R(G)$ may consist of approximate values of a graph parameter such as the diameter or the conductance of G. (Other examples are the classes of problems $\mathcal{BL}_{k,\epsilon}(\cdot)$ and $\mathcal{RP}_{k,\epsilon}(\cdot)$ considered in Section 4.2.) This notion of search problems extends the standard notion of decision problems for graphs (a.k.a graph properties).[10] The natural promise problem version of this notion couples such search problems with promises that are graph properties (i.e., G satisfies the promise P if and only if each graph that is isomorphic to G satisfies P).

Definition 4.6 (search promise problem for graphs): *A* search promise problem for graphs (for short, graph problem) *is a pair (P, R) such that P is a graph property and R is closed under isomorphism in the sense that if G and G' are isomorphic, then $R(G) = R(G')$.*

For a graph $G = ([N], E)$ and a set of vertices $S \subseteq [N]$, we denote by G_S the subgraph of G induced by S. By $\binom{[N]}{k}$ we denote the set of all k-subsets of $[N]$.

Theorem 4.7 *Let $\Pi = (P, R)$ be a search promise problem for graphs. Then, Π has a pseudodeterministic algorithm of query complexity q if and only if there exists k that is polynomially related to q and a function f such that for every $G = ([N], E) \in P$ there exists a $y \in R(G)$ such that $\Pr_{S \in \binom{[N]}{k}}[f(N, G_S) = y] \geq 2/3$. In particular, if Π has a pseudodeterministic algorithm then $k = 18q$ will do, whereas if such a k and f exist then Π has a pseudodeterministic algorithm of query complexity $q = \binom{k}{2}$.*

The function f associates a candidate solution with each k-vertex graph such that a large fraction of all induced k-vertex subgraphs of each $G \in P$ are associated with the same candidate solution (which is in $R(G)$). The search problem Π has a poly(k)-query pseudodeterministic algorithm if and only if such a function f exists.

Proof: Suppose that there exists k and a function f such that for every $G = ([N], E) \in P$ there exists a $y \in R(G)$ such that $\Pr_{S \in \binom{[N]}{k}}[f(N, G_S) = y] \geq 2/3$. Then, a pseudodeterministic

[10] One may also consider search problems of a different type in which solutions are sets of vertices and it is required that $S \in R(G)$ if and only if $\{\pi(v) : v \in S\}$ is a solution to the graph $\pi(G)$ (i.e., the relabelling of G under the permutation π). For example, $R(G)$ may consist of all triples of vertices that form a triangle in G. For further discussion, see the end of this section.

algorithm of query complexity $q = \binom{k}{2}$ is obtained by just sampling k vertices at random in the graph, obtaining the induced subgraph, denoted G', and outputting $f(N, G')$.

To prove the opposite direction, we follow the transformation[11] of [12, Sec. 4], while observing that its steps are still applicable. Loosely speaking, in [12, Sec. 4] it is shown that any q-query property tester (in the adjacency matrix model) can be transformed into one that takes a random sample of $2q$ vertices, queries the induced subgraph, and decides based on that subgraph. Here we show that this transformation applies also in the case of pseudodeterministic algorithms for graph problems (as in Definition 4.6). The point is that a canonical solution that is output with probability at least 2/3 can be treated as a decision bit. The following steps correspond to the transformation of [12, Sec. 4] (and the specific pointers are included as a source of additional details).

Step 1: Obtaining a vertex-uncovering algorithm (first part of [12, Lem. 4.1]).

A vertex-uncovering algorithm proceeds in iterations such that in each iteration a new vertex v is selected (possibly based on prior answers) and queries of the form (v, u) are made for each u that was selected in prior iterations. The transformation here is rather generic. Each query (v_1, v_2) is emulated by two iterations of the vertex-uncovering algorithm, which means that the query complexity may get squared. Actually it is more important to note that the number of vertices selected, denoted k_1, is at most twice the original query complexity.

Step 2: Obtaining an algorithm that inspects a random induced subgraph (second part of [12, Lem. 4.1]).

This step consists of observing what happens when we let the algorithm query a random isomorphic copy of the input graph. The original analysis relies on the fact that the algorithm's decision should remain valid also in this case, and this reasoning holds in our setting as well. Specifically, if the pseudodeterministic algorithm outputs y (with probability at least 2/3) as canonical solution for the graph $G \in P$, then the algorithm must output y with probability at least 2/3 when given an isomorphic copy of G.

Step 3: Obtaining a sample-independent output [12, Sec. 4.2.1].

In the context of [12], the output is a decision, but again the analysis generalizes. The issue at hand is that the original algorithm selects a random k_1-set of vertices S uniformly, and given that it saw the induced subgraph α, it outputs y with probability $q_{S,\alpha}(y)$. Instead, when seeing the induced subgraph α, the new algorithm outputs y with probability $q_\alpha(y) \stackrel{\text{def}}{=} \mathrm{E}_S[q_{S,\alpha}(y)]$, which is independent of the selected sample S. Note that if $G \in P$, then there exists a y such that $\mathrm{E}_S[q_{S,\pi(G)_S}(y)] \geq 2/3$, for every relabeling π of the graph G. Noting that $\pi(G)_S = G_{\pi(S)}$, for a random permutation π, it holds that $\mathrm{E}_{S,\pi}[q_{S,G_{\pi(S)}}(y)] \geq 2/3$, which implies that $\mathrm{E}_{S,S'}[q_{S,G_{S'}}(y)] \geq 2/3$. But this means that $\mathrm{E}_{S'}[q_{G_{S'}}(y)] \geq 2/3$, which means that the new algorithm satisfies the pseudodeterministic requirement.

Step 4: Obtaining an isomorphism-oblivious output [12, Sec. 4.2.2].

Step 3 established that the output depends only on the induced subgraph seen, and here we claim that it is oblivious of the labeling of this subgraph. The analysis is similar to the one underlying Step 3.

[11] This transformation of [12, Sec. 4] is often referred to as a "canonization" procedure, yielding a "canonical" tester. But this notion of "canonical" has nothing to do with the notion of canonical solutions.

Step 5: Obtaining an output that is determined by the induced subgraph [12, Sec. 4.2.3].

Step 4 established that the output depends only on the unlabeled induced subgraph, but this dependence may take a probabilistic form; that is, seeing a subgraph H causes the algorithm to output y with probability $p_H(y)$. Our goal is to have the output be uniquely determined as a function of the (unlabeled) induced subgraph. Following [12], we assume that the algorithm has error probability less than 1/6 (i.e., the probability that it outputs the canonical solution is at least 5/6). This assumption can be enforced by error reduction (but the number of vertices increases from k_1 to $9k_1$). At this point, we introduce a new algorithm that, upon seeing the subgraph H, outputs y if $p_H(y) > 1/2$ (and outputs nothing if $\max_y\{p_H(y)\} \leq 1/2$). Since the original algorithm outputs canonical solutions with probability at least 5/6, for every $G \in P$, it holds that $\mathrm{E}_S[p_{G_S}(y)] \geq 5/6$ for the canonical solution $y \in R(G)$. Therefore, $\Pr_S[p_{G_S}(y) > 1/2] \geq 2/3$. Define $f_N(H) = y$ if $p_H(y) > 1/2$ (and $f_N(H) = \perp$ if no such y exists).

We now obtain the desired function f: For any k-vertex graph H, define $f(N, H) = f_N(H)$. By Step 5, we know that for every $G = ([N], E) \in P$ there exists a $y \in R(G)$ such that $\Pr_{S\in\binom{[N]}{k}}[f(N, G_S) = y] \geq 2/3$. The theorem follows. ■

A different type of graph problems. In continuation to Footnote 10, we stress that Definition 4.6 only covers one type of graph problems; that is, graph problems for which solutions are values that are invariant under isomorphism of the input graph. In contrast, one may consider a more general type of graph problems, which we call labeled graph problems. In these problems solutions are triples (v, S, f) where v is a value (as in Definition 4.6), S is a set of vertices and $f : S \to \{0,1\}^*$ such that the following ("graph nature") condition holds: *If (v, S, f) is a solution for graph G, then $(v, \{\pi(u) : u \in S\}, f \circ \pi^{-1})$ is a solution for the graph that results from G by applying the vertex relabeling π.* A natural question about such labeled graph problems is whether they have pseudodeterministic algorithms of complexity that is smaller than that of deterministic algorithms.

5 Safely finding canonical solutions

An even stronger notion than finding canonical solutions is finding such solutions without ever outputting an alternative solution (but rather allowing a small probability of having no output).

Definition 5.1 (safely finding canonical solutions): *Let R, $R(x)$ and S_R be as in Definition 2.1. A* (possibly probabilistic) *algorithm A* safely finds canonical solutions with respect to *R if there exists a function $s : \{0,1\}^* \to \{0,1\}^*$ such that the following conditions hold:*

1. *As in Definition 2.1, for every $x \in S_R$ it holds that $s(x) \in R(x)$, whereas $s(x) = \lambda$ for every $x \in \{0,1\}^* \setminus S_R$.*
2. *For every $x \in \{0,1\}^*$, it holds that $\Pr[A(x)=s(x)] \geq 2/3$ and $A(x) \in \{s(x), \perp\}$ always holds.*

Definition 5.1, which is implicit in Gat's thesis [4], is robust in the sense that replacing the threshold 2/3 by either $1/\mathrm{poly}(|x|)$ or $1-\exp(-\mathrm{poly}(|x|))$ yields the same class (via standard error reduction). Indeed, this class of search problems is reminiscent of the class $\mathcal{ZPP}$ of decision problems recognized with zero-error probability. In fact:

Theorem 5.2 (Theorem 2.2, adapted): *A relation R has a probabilistic polynomial-time algorithm that safely finds canonical solutions w.r.t it if and only if finding solutions w.r.t R is P-reducible to some set in $\mathcal{ZPP}$.*

(The proof of Theorem 5.2 mimics the proof of Theorem 2.2.) Thus, the difference between finding canonical solutions and safely finding canonical solutions is reflected in the difference between $\mathcal{BPP}$ and $\mathcal{ZPP}$. This fact becomes even more evident when considering the set associated with any canonical solution function $s : \{0,1\}^* \to \{0,1\}^*$ for R. Specifically, let $S_s = \{(x,i,b) : s(x)_i = b\}$, where $s(x)_i$ denotes the i^{th} bit of $s(x)$. The actual contents of the proofs of Theorems 2.2 and 5.2 is captured by the following proposition.

Proposition 5.3 *A relation R has a probabilistic polynomial-time algorithm that finds* (resp., safely finds) *canonical solutions w.r.t it if and only if there exists a canonical solution function $s : \{0,1\}^* \to \{0,1\}^*$ for R such that the corresponding set S_s is in $\mathcal{BPP}$* (resp., in $\mathcal{ZPP}$).

Indeed, R has a deterministic polynomial-time algorithm that (safely) finds (canonical) solutions w.r.t it if and only if S_s is in $\mathcal{P}$.

Proof: The first direction (from S_s to finding s-canonical solutions) is proved by using the standard P-reduction of computing s to deciding S_s; that is, on input x, for all $i \leq \text{poly}(|x|)$ and $b \in \{0,1\}$, make the query (x,i,b) and constract $s(x)$ accordingly.[12] In particular, when using a ZPP algorithm for S_s and obtaining a failure notification, the search algorithm outputs $\perp$. For the opposite direction (i.e., from finding s-canonical solutions to S_s), recall that deciding S_s is P-reducible to computing s, which is turn is done by an algorithm that finds s-canonical solutions w.r.t R. Likewise, when using a safe finder, a failure notification makes the decider issue a similar notification. ■

Appendix: Sublinear-time pseudodeterministic algorithms related to LDC

Recall that local decodable codes (LDC) are codes for which each message bit can be correctly recovered (with probability 2/3) from a corrupted codeword (at error rate $\delta/3$) by making few queries to that codeword. We consider codes that stretch k-bit messages into n-bit codewords, which are at relative distance $\delta > 0$. Two concrete proposals follow.

1. Recall that for $n = \text{poly}(k)$ local decodability is feasible with poly-logarithmic many queries (and some constant $\delta > 0$). Let $\ell = \text{poly}(\log n)$. Then, considering only strings $w \in \{0,1\}^n$ that are $\delta/3$-close to the code, we define a relation R such that $y \in R(w)$ if y is an ℓ-bit (consecutive) substring of x and w is $\delta/3$-close to an encoding of x.

   Canonical solutions w.r.t this relation can be found probabilistically by making poly-logarithmically many queries. For example, by invoking the local decoder ($O(\log\log n)$ times) to recover each bit position $i \in \{1,\ldots,\ell\}$. Indeed, we have intensionally defined R in a way that avoids having unique solutions.

[12] As in the proof of Theorem 2.2, the oracle S_s is implemented by a BPP (resp., ZPP) algorithm of sufficiently low error probability.

2. Recall that for $n = 2^k$ local decodability is feasible with a constant number of queries (and $\delta = 1/2$). Indeed, this refers to the Hadamard code. Then, considering only strings that are 0.24-close to the code, we define a relation R such that $y \in R(w)$ if $w \in \{0,1\}^n$ is 0.24-close to an encoding of y. Canonical solutions w.r.t this relation can be found probabilistically by making logarithmically many queries (indeed, the double-logarithmic overhead involved in error reduction can be avoided; see [6, Sec. 2.5.2, Fn. 11]).[13]

   Indeed, as defined, R has unique solutions.

Both search problems can be generalized so that the sought solutions are not the strings denoted y, but rather search problems regarding y; that is, for any R as above and any search problem R', consider composed search problem R'' such that $z \in R''(w)$ if $y \in R(w)$ and $z \in R'(y)$. Indeed, if canonical solutions can be found w.r.t R' (by a standard probabilistic algorithm), then canonical solutions w.r.t R'' can be found in a number of queries that equals the number of queries needed for finding canonical solutions w.r.t R.

References

[1] Laszlo Babai and Eugene M. Luks. Canonical Labeling of Graphs. In *15th STOC*, pages 171–183, 1983.

[2] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar, Sampling Algorithms: Lower Bounds and Applications. In *33rd STOC*, pages 266–275, 2001.

[3] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, Vol. 288(1), pages 21–43, 2002.

[4] Eran Gat. On the canonization of probabilistic algorithms. M.Sc. Thesis, Weizmann Institute of Science, 2009.

[5] Eran Gat and Shafi Goldwasser. Probabilistic Search Algorithms and their Cryptographic Applications. ECCC TR11-136, 2011.

[6] Oded Goldreich. *Foundations of Cryptography – Basic Tools*. Cambridge University Press, 2001.

[7] Oded Goldreich. In a World of P=BPP. ECCC TR10-135, 2010.

[8] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, pages 653–750, July 1998. Extended abstract in *37th FOCS*, 1996.

[9] Oded Goldreich, Michael Krivelevich, Ilan Newman, and Eyal Rozenberg. Hierarchy Theorems for Property Testing. *Computational Complexity*, Vol. 21 (1), pages 129–192, 2012.

[10] Oded Goldreich and Dana Ron. Property Testing in Bounded Degree Graphs. *Algorithmica*, Vol. 32 (2), pages 302–343, 2002. Extended abstract in *29th STOC*, 1997.

[13] The idea is to self-correct $O(k)$ positions in the codeword, where these positions correspond to an auxiliary good (and linear) error correcting code, and capitalize on the fact that it suffices to correctly recover each such auxiliary bit with constant probability.

[11] Oded Goldreich and Dana Ron. A Sublinear Bipartitness Tester for Bounded Degree Graphs. *Combinatorica*, Vol. 19 (3), pages 335–373, 1999.

[12] Oded Goldreich and Luca Trevisan. Three theorems regarding testing graph properties. *Random Structures and Algorithms*, Vol. 23 (1), pages 23–57, August 2003.

[13] Hendrik Lenstra and Bart de Smit. Standard Models for Finite Fields. Lecture in *Foundations of Computational Mathematics*, 2008. Slides avialable from `http://www.damtp.cam.ac.uk/user/na/FoCM/FoCM08/Talks/Lenstra.pdf`

[14] Noam Nisan. CREW PRAMs and and decision trees. *SIAM Journal on Computing*, Vol. 20(6), pages 999–1007, 1991. Extended abstract in *21st STOC*, 1989.

[15] Michael Saks and Avi Wigderson. Probabilistic Boolean decision trees and the complexity of evaluation game trees. In proceedings of *27th FOCS*, pages 29–38, 1986.

[16] Marc Snir. Lower Bounds on Probabilistic Linear Decision Trees. *Theor. Comput. Sci.*, Vol. 38, pages 69–82, 1985.