Pseudo-Deterministic Proofs

Shafi Goldwasser[1], Ofer Grossman[2], and Dhiraj Holden[3]

1 **MIT, Cambridge MA, USA**
shafi@theory.csail.mit.edu
2 **MIT, Cambridge MA, USA**
ofer.grossman@gmail.com
3 **MIT, Cambridge MA, USA**
dholden@mit.edu

Abstract

We introduce *pseudo-deterministic interactive proofs* (psdIP): interactive proof systems for search problems where the verifier is guaranteed with high probability to output the same output on different executions. As in the case with classical interactive proofs, the verifier is a probabilistic polynomial time algorithm interacting with an untrusted powerful prover.

We view pseudo-deterministic interactive proofs as an extension of the study of pseudo-deterministic randomized polynomial time algorithms: the goal of the latter is to *find* canonical solutions to search problems whereas the goal of the former is to *prove* that a solution to a search problem is canonical to a probabilistic polynomial time verifier. Alternatively, one may think of the powerful prover as aiding the probabilistic polynomial time verifier to find canonical solutions to search problems, with high probability over the randomness of the verifier. The challenge is that pseudo-determinism should hold not only with respect to the randomness, but also with respect to the prover: a malicious prover should not be able to cause the verifier to output a solution other than the unique canonical one.

The $IP = PSPACE$ characterization implies that psdIP $= IP$. The challenge is to find constant round pseudo-deterministic interactive proofs for hard search problems. We show a constant round pseudo-deterministic interactive proof for the graph isomorphism problem: on any input pair of isomorphic graphs (G_0, G_1), there exist a unique isomorphism ϕ from G_0 to G_1 (although many isomorphism many exist) which will be output by the verifier with high probability, regardless of any dishonest prover strategy. In contrast, we show that it is unlikely that psdIP proofs with constant rounds exist for NP-complete problems by showing that if any NP-complete problem has a constant round psdIP protocol, then the polynomial hierarchy collapses.

1998 ACM Subject Classification F.1.1. Models of Computation (Probabilistic Algorithms)

Keywords and phrases Pseudo-Determinism, Interactive Proofs

Digital Object Identifier 10.4230/LIPIcs.ITCS.2018.17

1 Introduction

In [6], Gat and Goldwasser initiated the study of probabilistic (polynomial-time) search algorithms that, with high probability, output the same solution on different executions. That is, for all inputs x, the randomized algorithm A satisfies $Pr_{r_1,r_2}(A(x,r_1) = A(x,r_2)) \geq 1 - 1/poly(n)$.

Another way of viewing such algorithms is that for a fixed binary relation R, for every x the algorithm associates a canonical solution $s(x)$ satisfying $(x, s(x)) \in R$, and on input x the algorithm outputs $s(x)$ with overwhelmingly high probability. Algorithms that satisfy this condition are called *pseudo-deterministic*, because they essentially offer the same functionality as deterministic algorithms; that is, they produce a canonical output for each possible input (except with small error probability)[1]. 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, the output may arbitrarily distributed among all valid solutions for x (e.g. it may be uniformly distributed).

Several pseudo-deterministic algorithms have been found which improve (sometimes significantly) on the corresponding best known deterministic algorithm. This is the case for finding quadratic non-residues modulo primes, generators for certain cyclic groups, non-zeros of multi-variate polynomials, matchings in bipartite graphs in RNC, and sub-linear algorithms for several problems [8, 10, 6, 13]. For other problems, such as finding unique primes of a given length, pseudo-deterministic algorithms remain elusive (for the case of primes, it has been shown that there exists a subexponential time pseudo-deterministic algorithm which works on infinitely many input sizes [22]).

In this work we extend the study of pseudo-determinism in the context of probabilistic algorithms to the context of interactive proofs and non-determinism. We view pseudo-deterministic interactive proofs as a natural extension of pseudo-deterministic randomized polynomial time algorithms: the goal of the latter is to *find* canonical solutions to search problems whereas the goal of the former is to *prove* that a solution to a search problem is canonical to a probabilistic polynomial time verifier. This naturally models the cryptographic setting when an authority generates system-wide parameters (e.g. an elliptic curve for all to use or a generator of a finite group) and it must prove that the parameters were chosen properly.

1.1 Our Contribution

Consider the search problem of finding a large clique in a graph. A nondeterministic efficient algorithm for this problem exists: simply guess a set of vertices C, confirm in polynomial time that the set of vertices forms a clique, and either output C or reject if C is not a clique. Interestingly, in addition to being nondeterministic, there is another feature of this algorithm; on the same input graph there may be many possible solutions to the search problem and any one of them may be produced as output. Namely, on different executions of the algorithm, on the same input graph G, one execution may guess clique C and another execution may guess clique $C' \neq C$, and both are valid accepting executions.

A natural question is whether for each graph with a large clique, there exists a unique canonical large clique C which can be verified by a polynomial time verifier: that is, can the verifier V be convinced that the clique C is the canonical one for the input graph? Natural candidates which come to mind, such as being the lexicographically smallest large clique, are not known to be verifiable in polynomial time (but seem to require the power of Σ_2 computation).

In this paper, we consider this question in the setting of interactive proofs and ask whether the interactive proof mechanism enables provers to convince a probabilistic verifier of the "uniqueness of their answer" with high probability.

[1] In fact, by amplifying the success probability, one can ensure that as black boxes, pseudo-deterministic algorithms are indistinguishable from deterministic algorithms by a polynomial time machine.

Pseudo-deterministic Interactive Proofs: We define *pseudo-deterministic interactive proofs* for a search problem R (consisting of pairs $(instance, solution)$) with associated function s as a pair of interacting algorithms: a probabilistic polynomial time verifier and a computationally unbounded prover which on a common input instance x engage in rounds of interaction at the end of which with high probability the verifier output a canonical solution $y = s(x)$ for x if any solution exists and otherwise rejects x. Analogously to the case of completeness in interactive proofs for languages, we require *Canonical Completeness*: that for every input x, there exists an honest prover which can send the correct solution $s(x)$ to the verifier when one exists. Analogously to the case of soundness, we require *Canonical Soundness*: no dishonest prover can cause the verifier to output a solution other than $s(x)$ (the canonical one) (except with very low probability).

One may think of the powerful prover as aiding the probabilistic polynomial time verifier to find canonical solutions to search problems, with high probability over the randomness of the verifier. The challenge is that pseudo-determinism should hold not only with respect to the randomness, but also with respect to the prover: a malicious prover should not be able to cause the verifier to output a solution other than the canonical unique one. In addition to the intrinsic complexity theoretic interest in this problem, *consistency* or *predictability* of different executions on the same input are natural requirements from protocols.

We define *pseudo-deterministic IP* (psdIP) to be the class of search problems R (relation on inputs and solutions) for which there exists a pseudo-deterministic polynomial round interactive proof for R.

Theorem: For any problem L in NP, there is a pseudo-deterministic polynomial round interactive proof for the search problem R consisting of all pairs (x, w) where $x \in L$ and w is a witness for x.

One can prove the above theorem by noting that finding the lexicographically first witness w for x is a problem in PSPACE. Then, since IP = PSPACE [24], we know an interactive proof for finding the lexicographically first witness w exists. More formally we have:

Proof: Let us consider the function $f(x)$ which outputs the lexicographically first witness that $x \in L$ if $x \in L$ or $\perp$ otherwise. It is easy to see that determining whether $f(x) = y$ is in PSPACE. As a result, there is an polynomial-round IP protocol to determine whether $f(x) = y$. Then, the psdIP protocol is as follows; the prover gives the verifier y and then they run the protocol, and the verifier accepts and outputs y if the protocol accepts. This satisfies the conditions for pseudo-determinism because of the completeness and soundness properties of the IP protocol.

In light of the above, we ask: do **constant-round** *pseudo-deterministic interactive proofs exist for hard problems in* NP *for which many witnesses exist?* We let psdAM refer to those pseudo-deterministic interactive proofs in which a constant number of rounds is used. [2].

Graph Isomorphism is in pseudo-deterministic AM: Theorem 7: *There exists a pseudo-deterministic constant-round Arthur-Merlin protocol for finding an isomorphism between two given graphs.*

Recall that the first protocol showing graph non-isomorphism is in constant round IP was shown by [9] and later shown to be possible using public coins via the general transformation

[2] We note that historically, the class AM referred to protocols in which the verifiers' messages consisted of his coin tosses, namely public-coin protocols. In this work, we use AM to refer to constant round interactive proofs

of private to public coins [11]. Our algorithm finds a unique isomorphism by producing the lexicographically first isomorphism. In order to prove that a particular isomorphism between input graph pairs is lexicographically smallest, the prover will prove in a sequence of sub-protocols to the verifier that a sequence of graphs suitably defined are non-isomorphic. In an alternative construction, we exhibit an interactive protocol that computes the automorphism group of a graph in a verifiable fashion.

SAT is not in pseudo-deterministic AM: Theorem 9: *if any* NP*-complete problem has a a pseudo-deterministic constant round* AM *protocol, then,* NP $\subseteq$ coNP/*poly and the polynomial hierarchy collapses to the third level, showing that it is unlikely that NP complete problems have pseudo-deterministic constant round* AM *protocols.*

This result extends the work of [16] which shows that if there are polynomial time unique verifiable proofs for SAT, then the polynomial hierarchy collapses. Essentially, their result held for deterministic interactive proofs (i.e., NP), and we extend their result to probabilistic interactive proofs with constant number of rounds (i.e., AM).

Every problem in search-BPP is in subexponential-time pseudo-deterministic MA: Theorem 13: *For every problem in search-*BPP*, there exists a pseudo-deterministic* MA *protocol where the verifier takes subexponential time on infinitely many input lengths.*

The idea of the result is to use known circuit lower bounds to get pseudo-deterministic subexponential time MA protocols for problems in search-BPP for infinitely many input lengths. We remark that recently Oliveira and Santhanam [22] showed a subexponential time pseudo-deterministic algorithm for infinitely many input lengths for all properties which have inverse polynomial density and are testable in probabilistic polynomial time. (An example of such a property is the property of being prime, as the set of primes has polynomial density.) In their construction, the condition of high density is required in order for the property to intersect with their subexponential-size hitting set. (Subsequent work in [17] also drops this requirement but only results in an average-case pseudo-deterministic algorithm.) In the case of MA, unconditional circuit lower bounds for MA with a verifier which runs in exponential time have been shown by Miltersen et al [20], which allows us to no longer require inverse polynomial density. Hence, we can obtain a pseudo-deterministic MA algorithm from circuit lower bounds. Thus, compared to [22], our result shows a pseudo-derandomization (for a subexponential verifier and infinitely many input sizes n) for all problems in search-BPP (and not just those with high density), but requires a prover.

Pseudo-deterministic NL equals search-NL: Theorem 15: *For every search problem in search-*NL*, there exists a pseudo-deterministic* NL *protocol.*

We define *pseudo-deterministic NL* to be the class of search problems R (a relation on inputs and solutions) for which there exists log-space non-deterministic algorithm M (Turing machines) such that for every input x, there exists a unique $s(x)$ such that $R(x, s(x)) = 1$ and $M(x)$ outputs $s(x)$ or rejects x. Namely, there are no two accepting paths for input x that result in different outputs.

To prove the above theorem, we look at the problem of directed connectivity (that is, given a directed graph G with two vertices s and t, we find a path from s to t), and we show that it is possible to find the lexicographically first path of shortest length in NL. To do so, we first find the length d of the shortest path, which can be done in NL. Then, we find the lexicographically first outneighbor u of s such that there is a path of length $d - 1$ from u to t. This can be done by going in order over all outneighbors of s, and for each of them checking if there is a path of length $d-1$ to t (if there is not such a path, that can be demonstrated in NL since NL = coNL [18, 25]). By recursively applying this protocol to find a path from u to t, we end up obtaining the lexicographically first path of shortest length, which is unique.

Structural Results: We show a few structural results regarding pseudo-deterministic interactive proofs In Section 7. Specifically, we show that psdAM equals to the class search$-\mathrm{P}^{\mathrm{promise}-(\mathrm{AM}\cap\mathrm{coAM})}$, where for valid inputs x, all queries to the oracle must be in the promise. We show similar results in the case of pseudo-deterministic MA and pseudo-deterministic NP.

1.2 Other Related Work

In their seminal paper on NP with unique solutions, Valiant and Vazirani asked the following question: is the inherent intractability of NP-complete problems caused by the fact that NP-complete problems have many solutions? They show this is not the case by exhibiting a problem – SAT with unique solutions – which is NP-hard under randomized reductions. They then showed how their result enables to show the NP-hardness under randomized reductions for a few related problems such as parity-SAT. We point out that our question is different. We are not restricting our study to problems (e.g. satisfiable formulas) with unique solutions. Rather, we consider hard problems for which there may be exponentially many solutions, and ask if one can focus on one of them and verify it in polynomial time. In the language of satisfiability, ϕ can be any satisfiable formula with exponentially many satisfying assignments; set $s(\phi)$ to be a unique valued function which outputs a satisfying assignment for ϕ. We study whether there exists an s which can be efficiently computed, or which has an efficient interactive proof.

The question of computing canonical labellings of graphs was considered by Babai and Luks [3] in the early eighties. Clearly graph isomorphism is polynomial time reducible to computing canonical labellings of graphs (compute the canonical labeling for your graphs and compare), however it is unknown whether the two problems are equivalent (although finding canonical labellings in polynomial time seems to be known for all classes of graphs for which isomorphism can be computed in polynomial time). The problem of computing a set of generators (of size $O(\log n)$) of the automorphism group of a graph G was shown by Mathon [19] (among other results) to be polynomial-time reducible to the problem of computing the isomorphism of a graph. We use this in our proof that graph isomorphism is in psdAM.

A line of work on search vs decision and hierarchy collapses, some in the flavor of our result of Section 4, have appeared in [16, 15, 14, 4].

Finally, we mention that recently another notion of uniqueness has been studied in the context of interactive proofs by Reingold et al [23], called *unambiguous interactive proofs* where the prover has a unique successful strategy. This again differs from pseudo-deterministic interactive proofs, in that we don't assume (nor guarantee) a unique strategy by the successful prover, we only require that the prover proves that the solution (or witness) the verifier receives is unique (with high probability).

1.3 Subsequent Work

In [17], inspired by this work, Holden shows that for every BPP search problem there exists an algorithm A which for infinitely many input lengths n and for every polynomial-time samplable distribution over inputs of length n runs in subexponential time and produces a unique answer with high probability on inputs drawn from the distribution and over A's random coins.

[17] expands on the work of Oliveira and Santhanam [22] in several ways. Whereas the latter give a pseudo-deterministic algorithm for estimating the acceptance probability of a circuit on inputs of a given length, the former applies to general search-BPP problems, where the input is a string of a given length over some alphabet and algorithm's A goal is to output a solution that satisfies a BPP testable relation with the input string. Holden [17] shows that for infinitely many input lengths, average-case (over the input distribution) pseudo-deterministic algorithms are possible for problems in search-BPP.

2 Definitions of Pseudo-deterministic Interactive Proofs

In this section, we define pseudo-determinism in the context of nondeterminism and interactive proofs. We begin by defining a search problem.

▶ **Definition 1** (Search Problem). A *search problem* is a relation R consisting of pairs (x, y). We define L_R to be the set of x's such that there exists a y satisfying $(x, y) \in R$. An algorithm solving the search problem is an algorithm that, when given $x \in L_R$, finds a y such that $(x, y) \in R$. When L_R contains all strings, we say that R is a *total* search problem. Otherwise, we say R is a *promise* search problem.

We now define pseudo-determinism in the context of interactive proofs for search problems. Intuitively speaking, we say that an interactive proof is pseudo-deterministic if an honest prover causes the verifier to output the same unique solution with high probability (canonical completeness), and dishonest provers can only cause the verifier to output either the unique solution or $\perp$ with high probability (canonical soundness). In other words, dishonest provers cannot cause the verifier to output an answer which is not the unique answer. Additionally, we have the condition that for an input x with no solutions, for all provers the verifier will output $\perp$ with high probability (standard soundness). We note that we use psdIP, psdAM, psdNP, psdMA, and so on, to refer to a class of promise problems, unless otherwise stated.

▶ **Definition 2** (Pseudo-deterministic IP). A search problem R is in *pseudo-deterministic* IP (often denoted psdIP) if there exists a function s mapping inputs to the search problem to solutions (i.e., all $x \in L_R$ satisfy $(x, s(x)) \in R$), and there is an interactive protocol between a probabilistic polynomial time verifier algorithm V and a prover (unbounded algorithm) P such that for every $x \in L_R$:

1. (Canonical Completeness) There exists a P such that $\Pr_r[(P, V)(x, r) = s(x)] \geq \frac{2}{3}$. (We use $(P, V)(x, r)$ to denote the output of the verifier V when interacting with prover P on input x using randomness r).
2. (Canonical Soundness) For all P', $\Pr_r[(P', V)(x, r) = s(x) \text{ or } \perp] \geq \frac{2}{3}$.

And (Standard Soundness) for every $x \notin L_R$, for all provers P', $\Pr_r[(P', V)(x, r) \neq \perp] \leq \frac{1}{3}$.

One can similarly define pseudo-deterministic MA, and pseudo-deterministic AM, where MA is a 1-round protocol, and AM is a 2-round protocol. One can show that any constant-round interactive protocol can be reduced to a 2-round interactive protocol [2]. Hence, the definition of pseudo-deterministic AM captures the set of all search problems solvable in a constant number of rounds of interaction.

Historical Note: Historically, AM referred to public coin protocols, whereas IP referred to private coin protocols. In this work, we use AM to refer to constant round protocols, and IP to refer to polynomial round protocols (unless explicitly stated otherwise). By the result of [11], we know that when the prover is all-powerful, a private coin protocol can be simulated using private coins, so in this setting the distinction between private and public coins does not matter.

▶ **Definition 3** (Pseudo-deterministic AM). A search problem R is in *pseudo-deterministic* AM (often denoted psdAM) if there exists a function s where all $x \in L_R$ satisfy $(x, s(x)) \in R$, a probabilistic polynomial time verifier algorithm V, and polynomials p and q, such that for every $x \in L_R$:

1. (Canonical Completeness) $\Pr_{r \in \{0,1\}^{p(n)}}(\exists z \in \{0,1\}^{q(n)}\, V(x,r,z) = s(x)) \geq \frac{2}{3}$
2. (Canonical Soundness) $\Pr_{r \in \{0,1\}^{p(n)}}(\forall z \in \{0,1\}^{q(n)}\, V(x,r,z) \in \{s(x), \perp\}) \geq \frac{2}{3}$.

And (Standard Soundness) for every $x \notin L_R$, we have $\Pr_{r \in \{0,1\}^{p(n)}}(\forall z \in \{0,1\}^{q(n)}\, V(x,r,z) = \{\perp\}) \geq \frac{2}{3}$.

▶ **Definition 4** (Pseudo-deterministic MA). A search problem R is in *pseudo-deterministic* MA (often denoted psdMA) if there exists a function s where all $x \in L_R$ satisfy $(x, s(x)) \in R$ and $|s(x)| \leq \mathit{poly}(x)$, a probabilistic polynomial time verifier V such that for every $x \in L_R$[3]:

1. (Canonical Completeness) There exists a message M of polynomial size such that $\Pr_r[V(x,M,r) = s(x)] \geq \frac{2}{3}$.
2. (Canonical Soundness) For all M', $\Pr_r[V(x,M',r) = s(x) \text{ or } \perp] > \frac{2}{3}$.

And (Standard Soundness) for every $x \notin L_R$, for all M', $\Pr_r[V(x,M',r) \neq \perp] \leq \frac{1}{3}$.

Pseudo-determinism can similarly be defined in the context of NP (which can be viewed as a specific case of an interactive proof):

▶ **Definition 5** (Pseudo-deterministic NP). A search problem R is in *pseudo-deterministic* NP (often denoted psdNP) if there exists a function s where all $x \in L_R$ satisfy $(x, s(x)) \in R$ and $|s(x)| \leq \mathit{poly}(x)$, and there is a deterministic polynomial time verifier V such that for every $x \in L_R$:

1. There exists a message M of polynomial size such that $V(x, M) = s(x)$.
2. For all M', $V(x, M') = s(x)$ or $V(x, M') = \perp$.

And for every $x \notin L_R$, for all M', we have $V(x, M') = \perp$.

A similar definition for pseudo-deterministic NL follows naturally:

▶ **Definition 6** (Pseudo-deterministic NL). A search problem R is in *pseudo-deterministic* NL (often denoted psdNL) if there exists a function s where all $x \in L_R$ satisfy $(x, s(x)) \in R$ and $|s(x)| \leq \mathit{poly}(x)$, there is a nondeterministic log-space machine V such that for every $x \in L_R$:

1. There exist nondeterministic choices N for the machine such that such that $V(x, N) = s(x)$.
2. For all possible nondeterministic choices N', $V(x, N') = s(x)$ or $V(x, N') = \perp$.

And for every $x \notin L_R$, for all nondeterministic choices N', $V(x, N') = \perp$.

[3] We remark that we use M to denote the proof sent by the prover Merlin, and not the algorithm implemented by the prover.

3 Pseudo-deterministic-AM algorithm for graph isomorphism

In this section we give an algorithm for finding an isomorphism between two graphs in AM that outputs the same answer with high probability. The way this algorithm works is that the prover will send the lexicographically first isomorphism to the verifier and then prove that it is the lexicographically first isomorphism. To prove that the isomorphism is the lexicographically first isomorphism, we label the graph and run a sequence of graph non-isomorphism protocols to show no lexicographically smaller isomorphism exists. We present an alternate proof of the same result in the appendix (the proof in the appendix is more group theoretic, whereas the proof below is more combinatorial).

▶ **Theorem 7.** *Finding an isomorphism between graphs can be done in* psdAM.

Proof. Let the vertices of G_1 be $v_1, v_2, \dots, v_n$, and the vertices of G_2 be $u_1, u_2, \dots, u_n$. We will show an AM algorithm which outputs a unique isomorphism ϕ. Our algorithm will proceed in n stages (which we will later show can be parallelized). After the kth stage, the values $\phi(v_1), \phi(v_2), \dots, \phi(v_k)$ will be determined.

Suppose that the values $\phi(v_1), \phi(v_2), \dots, \phi(v_k)$ have been determined. Then we will determine the smallest r such that there exists an isomorphism ϕ^* such that for $1 \leq i \leq k$, we have $\phi^*(v_i) = \phi(v_i)$, and in addition, $\phi^*(v_{k+1}) = u_r$. If we find r, we can set $\phi(v_{k+1}) = \phi^*(v_{k+1})$ and continue to the $k+1^{th}$ stage.

To find the correct value of r, the (honest) prover will tell the verifier the value of r and ϕ. Then, to show that the prover is not lying, for each $r' < r$, the prover will prove that there exists no isomorphism ϕ' such that for $1 \leq i \leq k$, we have $\phi'(v_i) = \phi(v_i)$, and in addition, $\phi'(v_{k+1}) = u_{r'}$. To prove this, the verifier will pick G_1 or G_2, each with probability 1/2. If the verifier picked G_1, he will randomly shuffle the vertices $v_{k+2}, \dots, v_n$, and send the shuffled graph to the prover. If the verifier picked G_2, he will set $u'_i = \phi(v_i)$ for $1 \leq i \leq k$, and $u'_{k+1} = u_{r'}$, and shuffle the rest of the vertices. If the prover can distinguish between whether the verifier initially picked G_1 or G_2, then that implies there is no isomorphism sending v_i to $\phi(v_i)$ for $1 \leq i \leq k$, and sending v_{k+1} to $u_{r'}$. The prover now can show this for all $r' \leq r$ (in parallel), as well as exhibit the isomorphism ϕ, thus proving that r is the minimum value such that there is an isomorphism sending v_i to $\phi(v_i)$ for $1 \leq i \leq k$, and sending v_{k+1} to u_r.

The above n stages can be done in parallel in order to achieve a constant round protocol. To do so, in the first stage, the prover sends the isomorphism ϕ to the verifier. Then, the verifier can test (in parallel) for each k whether under the assumption that $\phi(v_1), \phi(v_2), \dots, \phi(v_k)$ are correct, $\phi(v_{k+1})$ is the lexicographically minimal vertex which v_{k+1} can be sent to. The correctness of this protocol follows from the fact that multiple AM interactive proofs can be performed in parallel while maintaining soundness and completeness for all of the interactive proofs performed (as shown in appendix C.1 of [7]).

We note that in the above protocol, the prover only needs to have the power to solve graph isomorphism (and graph non-isomorphism). Also, we note that the above protocol uses private coins. While the protocol can be simulated with a public coin protocol [11], the simulation requires the prover to be very powerful. ◀

4 Lower bound on pseudo-deterministic AM algorithms

In this section, we establish that if any NP-complete problem has an AM protocol that outputs a unique witness with high probability, then the polynomial hierarchy collapses. To do this we show the analog of AM $\subseteq$ NP/*poly* for the pseudo-deterministic setting, and then use this fact to get a NP/*poly* algorithm with a unique witness. We can then use [16] to show that NP $\subseteq$ coNP/*poly*, which obtains the hierarchy collapse.

We begin by proving that psdAM $\subseteq$ psdNP/*poly*:

▶ **Lemma 8.** *Suppose that there is a* psdAM *protocol for a search problem R, which on input $x \in L_R$, outputs $s(x)$. Then, the search problem R has a* psdNP/*poly algorithm which, on input x, outputs $s(x)$.*

Proof. Consider a psdAM protocol, and suppose that on input $x \in L_R$, it outputs $s(x)$.

Since we are guaranteed that when the verifier of the the psdAM accepts, it will output $s(x)$ with high probability, we can use standard amplification techniques to show that the verifier will output $s(x)$ with probability $1 - o(\exp(-n))$, assuming an honest prover, and will output anything other than $s(x)$ with probability $o(\exp(-n))$, even with a malicious prover. Then, by a union bound, there exists a choice of random string r that makes the verifier output $s(x)$ for all inputs $x \in \{0,1\}^n$ of size n with an honest prover, and that for malicious provers, the verifier will either reject or output $s(x)$. We encode this string r as the advice string for the NP/*poly* machine.

The NP/*poly* machine computing s can read r off the advice tape and then guess the prover's message, and whenever the verifier accepts, $s(x)$ will be output by that nondeterministic branch. Thus $s(x)$ can be computed by an NP/*poly* machine. ◀

Next, we show that if an NP-complete problem has a pseudo-deterministic-NP/*poly* algorithm, then the polynomial hierarchy collapses.

▶ **Theorem 9.** *Let $L \in$ NP be an NP-complete problem. Let R be a polynomial time algorithm such that there exists a polynomial p so that $x \in L$ if and only if $\exists y \in \{0,1\}^{p(|x|)} R(x,y)$. Suppose that there is a* psdAM *protocol that when given some $x \in L$, outputs a unique $s(x) \in \{0,1\}^{p(|x|)}$ such that $R(x, s(x)) = 1$. Then,* NP $\subseteq$ coNP/*poly and the polynomial hierarchy collapses to the third level.*

Proof. Assume that there is a psdAM protocol that when given some $\phi \in L$, outputs a unique $s(\phi) \in \{0,1\}^{p(|\phi|)}$ such that $R(\phi, s(\phi)) = 1$. From Lemma 8, we have that there exists psdNP/*poly* algorithm A that given $\phi \in L$, outputs a unique witness $s(\phi)$ for ϕ. Given such an algorithm A, we can construct a function g computable in psdNP/*poly* that on two inputs ϕ_1 and ϕ_2, $g(\phi_1, \phi_2)$ is one of either ϕ_1 or ϕ_2 with the condition that if either ϕ_1 or ϕ_2 is in L, then $g(\phi_1, \phi_2)$ is satisfiable. If neither ϕ_1 nor ϕ_2 are in L, then $g(\phi_1, \phi_2) = \perp$.

To construct such a g, define a function g' where $g'(\phi_1, \phi_2) = \{\phi_1, \phi_2\} \cap L$ (i.e., $g'(\phi_1, \phi_2)$ is the subset of $\{\phi_1, \phi_2\}$ consisting of satisfiable formulas). We construct g by reducing the language $L' = \{(\phi_1, \phi_2) | g'(\phi_1, \phi_2) \neq \emptyset\}$ (which is in NP, and hence reducible to L, since L is NP-complete) to L and running A to find a unique witness for g, which we can then turn into a witness for L'. Note that a witness for L' is either a witness for ϕ_1 or for ϕ_2. We can then check whether this unique witness is a witness for ϕ_1 or ϕ_2, and output the ϕ_i for which it is a witness (in the case that the witness works for both of the ϕ_i, we output the lexicographically first ϕ_i).

We note that we view g as a function on the set $\{\phi_1, \phi_2\}$. That is, we set $g(\phi_1, \phi_2) = g(\phi_2, \phi_1)$ (if a function g does not satisfy this property, we can create a g^* satisfying this property by setting $g^*(\phi_1, \phi_2) = g(min(\phi_1, \phi_2), max(\phi_2, \phi_1))$).

Now, our goal is to use g, which we know is computable in psdNP/*poly* to construct an NP/*poly* algorithm for $\bar{L}$ (the complement of L).

We construct the advice string for L for length n as follows. Our advice string will be a set S consisting of strings ϕ_i. Start out with $S = \emptyset$. We know that there exists a $\phi_1 \in \{0,1\}^n \cap L$

such that $g(\phi, \phi_1) = x$ for at least half of the set $\{\phi \in \{0,1\}^n \cap L | g(\phi, s) = s \forall s \in S\}$. Such an s exists since in expectation, when picking a random s, half of the ϕ's will satisfy $g(\phi, s) = x$. If we keep doing this, we get a set S with $|S| \leq poly(n)$ such that for every $\phi \in L$ of length n, there exists an $s \in S$ such that $g(\phi, s) = x$.

Now, to check that $\phi \in \bar{L}$ in NP/*poly* (where S as defined above is the advice), we compute $g(\phi, s)$ for every $s \in S$, and check that $g(\phi, s) = s$ for every $s \in S$ which is possible because $|S|$ is polynomial in n. It is clear that this algorithm accepts if $\phi \notin L$ and rejects if $\phi \in L$, so therefore $L \in$ coNP/*poly*, which implies that NP $\subseteq$ coNP/*poly*. Furthermore, NP $\subseteq$ coNP/*poly* implies that the polynomial hierarchy collapses to the third level. ◀

5 Pseudo-deterministic derandomization for BPP in subexponential time MA

In this section, we prove the existence of pseudo-deterministic subexponential time (time $O(2^{n^\epsilon})$ for every ϵ) MA protocols for problems in search-BPP for infinitely many input lengths.

In this section, we prove that every problem R in search-BPP has an MA proof where the verifier takes subexponential time (and the prover is unbounded). For completeness, we define search-BPP below:

▶ **Definition 10** (Search-BPP)**.** A binary relation R is in *search-BPP* if there exist probabilistic polynomial-time algorithms A, B such that

1. Given $x \in R_L$, A outputs a y such that with probability at least 2/3, $(x, y) \in R$.
2. If y is output by A when run on x, and $(x, y) \notin R$, then B rejects on (x, y) with probability at least 2/3. Furthermore, for all $x \in L_R$, with probability at least 1/2 B accepts on (x, y) with probability at least 1/2.

When $x \notin R_L$, A outputs $\perp$ with probability at least 2/3.

The intuition of the above definition is that A is used to find an output y, and then B can be used to verify y, and amplify the success probability.

A main component of our proof will be the Nisan-Wigderson pseudo-random generator, which shows a way to construct pseudorandom strings given access to an oracle solving a problem of high circuit complexity.

To obtain the best running time for our pseudo-deterministic algorithm, we will need the iterated exponential functions first used in complexity theory by [20]. We will be considering functions with half-exponential growth, i.e. functions f such that $f(f(n)) \in O(2^{n^k})$ for some k.

▶ **Definition 11** (Fractional exponentials [20])**.** The fractional exponential function $e_\alpha(x)$ will be defined as $A^{-1}(A(x) + \alpha)$, where A is the solution to the functional equation $A(e^x - 1) = A(x) + 1$. In addition, we can construct such functions so that $e_\alpha(e_\beta(x)) = e_{\alpha+\beta}(x)$. It is clear from this definition that $e_1(n) = O(2^n)$, and that $e_{1/2}(e_{1/2}(x)) = O(2^n)$. We call a function f satisfying $f(x) = \Theta(e_{1/2}(x))$ a *half-exponential* function.

▶ **Definition 12** (Half-Exponential Time MA)**.** We define a *half-exponential time* MA *proof* to be an interactive MA proof in which the verifier runs in half-exponential time.

▶ **Theorem 13.** *Given a problem R in search-*BPP*, it is possible to obtain a pseudo-deterministic* MA *algorithm for R where the verifier takes subexponential time for infinitely many input lengths.*

Proof. From [20], we see that MA ∩ coMA where the verifier runs in half-exponential time cannot be approximated by polynomial-sized circuits. By Nisan-Wigderson [21], it follows that in half-exponential time MA, one can construct a pseudorandom generator with half-exponential stretch which is secure against any given polynomial-size circuit for infinitely many input lengths. We provide more details below.

Let T be the truth-table of a hard function in MA ∩ coMA. Then, let R be a relation in search-BPP. Recall from Definition 10 that there is an algorithm A that given x, produces y such that $(x, y) \in R$ with high probability if $x \in R_L$.

We will now describe the MA protocol. First, the prover sends T to the verifier and proves that it is indeed the truth table of the hard function in half-exponential time MA (which can be done in half-exponential time). With T in hand, the verifier can then compute the output of the Nisan-Wigderson pseudorandom generator. The verifier loops through the seeds in lexicographic order and uses the pseudorandom generator on each seed to create pseudo-random strings, which the verifier then uses as the randomness for A. Each time, the verifier tests whether $(x, A(x, r)) \in R$ (which can be done in BPP, and hence also in MA) and returns the first such valid output.

This will output the same solution whenever the verifier both gets the correct truth-table for the PRG, and succeeds in testing for each PRG output whether the output it provides is valid. Both of these happen with high probability, and thus this is a pseudo-deterministic subexponential-time MA algorithm for any problem in search-BPP which succeeds on infinitely many input lengths. ◀

6 Uniqueness in NL

In this section, we prove that every problem in search-NL can be made pseudo-deterministic. For completeness we include a definition of search-NL:

▶ **Definition 14** (search-NL). A search problem R is in *search*-NL if there is a nondeterministic log-space machine V such that for every $x \in L_R$,

1. There exist nondeterministic choices N for the machine such that such that $V(x, N) = y$, and $(x, y) \in R$.
2. For all possible nondeterministic choices N', $(x, V(x, N')) \in R$, or $V(x, N') = \bot$.

And for every $x \notin L_R$, for all nondeterministic choices N', $V(x, N') = \bot$.

▶ **Theorem 15** (Pseudo-determinism NL). *Every search problem in search-*NL *is in* psdNL.

One can think of the complete search problem for NL as: given a directed graph G, and two vertices s and t such that there is a path from s to t, find a path from s to t. Note that the standard nondeterministic algorithm of simply guessing a path will result in different paths for different nondeterministic guesses. Our goal will be to find a unique path, so that on different nondeterministic choices, we will not end up with a path which is not the unique one.

The idea will be to find the lexicographically first shortest path (i.e, if the min-length path from s to t is of length d, we will output the lexicographically first path of length d from s to t). To do so, first we will determine the length d of the min-length path from s to t. Then, for each neighbor of s, we will check if it has a path of length $d - 1$ to t, and move to the first such neighbor. Now, we have reduced the problem to finding a unique path of length $d - 1$, which we can do recursively.

The full proof is given below:

Proof. Given a problem in search-NL, consider the set of all min-length computation histories. We will find the lexicographically first successful computation history in this set.

To do so, we first (nondeterministically) compute the length of the min-length computation history. This can be done because coNL = NL (so if the shortest computation history is of size T, one can show a history of size T. Also, because it is coNL to show that there is no history of size up to $T-1$, we can show that there is no history of size less than T in NL).

In general, using the same technique, given a state S of the NL machine, we can tell what is the shortest possible length for a successful computation history starting at S.

Our algorithm will proceed as follows. Given a state S (which we initially set to be the initial configuration of the NL machine), we will compute T, the length of the shortest successful computation path starting at S. Then, for each possible nondeterministic choice, we will check (in NL) whether there exists a computation history of length $T-1$ given that nondeterministic choice. Then, we will choose the lexicographically first such nondeterministic choice, and recurse.

This algorithm finds the lexicographically first computation path of minimal length which is unique. Hence, the algorithm will always output the same solution (or reject), so the algorithm is pseudo-deterministic. ◀

7 Structural Results

In [8], Goldreich et al showed that the set of total search problems solved by pseudo-deterministic polynomial time randomized algorithms equals the set of total search problems solved by deterministic polynomial time algorithms, with access to an oracle to decision problems in BPP. In [10], this result was extended to the context of RNC. We show analogous theorems here. In the context of MA, we show that for total search problems, $\mathrm{psdMA} = \mathrm{search}{-}\mathrm{P}^{\mathrm{MA}\cap\mathrm{coMA}}$.[4] In other words, any pseudo-deterministic MA algorithm can be simulated by a polynomial time search algorithm with an oracle solving decision problems in $\mathrm{MA} \cap \mathrm{coMA}$, and vice versa.

In the case of search problems that are not total, we show that psdMA equals to the class $\mathrm{search}{-}\mathrm{P}^{\mathrm{promise}-(\mathrm{MA}\cap\mathrm{coMA})}$, where when the input x is in L_R, all queries to the oracle must be in the promise. We note that generally, when having an oracle to a promise problem, one is allowed to query the oracle on inputs not in the promise, as long as the output of the algorithm as a whole is correct for all possible answers the oracle gives to such queries. In our case, we simply do not allow queries to the oracle to be in the promise. Such reductions have been called *smart* reductions [12].

We show similar theorems for AM, and NP. Specifically, we show
$\mathrm{psdAM} = \mathrm{search}{-}\mathrm{P}^{\mathrm{promise}-(\mathrm{AM}\cap\mathrm{coAM})}$ and $\mathrm{psdNP} = \mathrm{search}{-}\mathrm{P}^{\mathrm{promise}-(\mathrm{NP}\cap\mathrm{coNP})}$, where the reductions to the oracles are smart reductions.

In the case of total problems, one can use a similar technique to show $\mathrm{psdAM} = \mathrm{search}{-}\mathrm{P}^{\mathrm{AM}\cap\mathrm{coAM}}$ and $\mathrm{psdNP} = \mathrm{search}{-}\mathrm{P}^{\mathrm{NP}\cap\mathrm{coNP}}$, where the oracles can only return answers to total decision problems.

▶ **Theorem 16.** *The class* psdMA *equals the class* $\mathrm{search}{-}\mathrm{P}^{\mathrm{promise}-(\mathrm{MA}\cap\mathrm{coMA})}$*, where on any input $x \in L_R$, the all queries to the oracle are in the promise.*

Proof. The proof is similar to the proofs in [8] and [10] which show similar reductions to decision problems in the context of pseudo-deterministic polynomial time algorithms and

[4] What we call search−P is often denoted as FP.

pseudo-deterministic NC algorithms.

First, we show that a polynomial time algorithm with an oracle for promise$-$(MA$\cap$coMA) decision problems which only asks queries in the promise has a corresponding pseudo-deterministic MA algorithm. Consider a polynomial time algorithm A which uses an oracle for promise$-$(MA $\cap$ coMA). We can simulate A by an MA protocol where the prover sends the verifier the proof for every question which A asks the oracle. Then, the verifier can simply run the algorithm from A, and whenever he accesses the oracle, he instead verifies the proof sent to him by the prover.

We note that the condition of a smart reduction is required in order for the prover to be able to send to the verifier the list of all queries A will make to the oracle. If A can ask the oracle queries not in the promise, it may be that on different executions of A, different queries will be made to the oracle (since A is a adaptive, and the queries A makes may depend on the answers returned by the oracle for queries not in the promise), so the prover is unable to predict what queries A will need answered.

We now show that a pseudo-deterministic MA algorithm B has a corresponding polynomial time algorithm A that uses a promise$-$(MA $\cap$ coMA) oracle while only querying on inputs in the promise. On input $x \in L_R$, the polynomial time algorithm can ask the promise$-$(MA $\cap$ coMA) oracle for the first bit of the unique answer given by B. This is a decision problem in promise$-$(MA $\cap$ coMA) since it has a constant round interactive proof (namely, run B and then output the first bit). Similarly, the algorithm A can figure out every other bit of the unique answer, and then concatenate those bits to obtain the full output.

Note that it is required that the oracle is for promise$-$(MA $\cap$ coMA), and not just for promise$-$MA, since if one of the bits of the output is 0, the verifier must be able to convince the prover of that (and this would require a promise$-$coMA protocol). ◀

A very similar proof shows the following:

▶ **Theorem 17.** *The class* psdNP *equals the class* search$-$P$^{\text{promise}-(\text{NP}\cap\text{coNP})}$*, where on any input $x \in L_R$, all queries to the oracle are in the promise.*

We now prove a similar theorem for the case of AM protocols. We note that this is slightly more subtle, since it's not clear how to simulate a search$-$P$^{\text{promise}-(\text{AM}\cap\text{coAM})}$ protocol using only a constant number of rounds of interaction, since the search-P algorithm may ask polynomial many queries in an adaptive fashion.

▶ **Theorem 18.** *The class* psdAM *equals the class* search$-$P$^{\text{promise}-(\text{AM}\cap\text{coAM})}$*, where on any input $x \in L_R$, the all queries to the oracle are in the promise.*

Proof. First, we show that a polynomial time algorithm with an oracle for promise$-$(AM $\cap$ coAM) decision problems where the queries are all in the promise has a corresponding pseudo-deterministic AM algorithm. We proceed similarly to the proof of Theorem 16. Consider a polynomial time algorithm A which uses an oracle for promise$-$(AM $\cap$ coAM). The prover will internally simulate that algorithm A, and then send to the verifier a list of all queries that A makes to the promise$-$(AM $\cap$ coAM) oracle. Then, the prover can prove the answer (in parallel), to all of those queries.

To prove correctness, suppose that the prover lies about at least one of the oracle queries. Then, consider the first oracle query to which the prover lied. Then, by a standard simulation argument, one can show that it can be made overwhelmingly likely that the verifier will discover that the prover lied on that query.

Once all queries have been answered by the verifier the algorithm B can run like A, but instead of querying the oracle, it already knows the answer since the prover has proved it to him.

The proof that a pseudo-deterministic MA algorithm B has a corresponding polynomial time algorithm A that uses an promise$-(\mathrm{AM} \cap \mathrm{coAM})$ oracle is identical to the proof of Theorem 16 ◀

As a corollary of the above, we learn that private coins are no more powerful than public coins in the pseudo-deterministic setting:

▶ **Corollary 19.** *A pseudo-deterministic constant round interactive proof using private coins can be simulated by a pseudo-deterministic constant round interactive proof using public coins.*

Proof. By Theorem 18, we can view the algorithm as an algorithm in search$-\mathrm{P}^{\mathrm{AM}\cap\mathrm{coAM}}$.

By a similar argument to that in Theorem 18, one can show that psdIP = search$-\mathrm{P}^{\mathrm{IP}\cap\mathrm{coIP}}$, where in this context IP refers to *constant* round interactive proofs using *private coins*, and AM refers to constant round interactive proofs using *public coins*. Since promise$-(\mathrm{AM}\cap\mathrm{coAM})$ = promise$-(\mathrm{IP} \cap \mathrm{coIP})$, since every constant round *private coin* interactive proof for decision problems can be simulated by a constant round interactive proof using *public coins* [11], we have:

$$\mathrm{psdAM} = \mathrm{search}{-}\mathrm{P}^{\mathrm{promise}-(\mathrm{AM}\cap\mathrm{coAM})} = \mathrm{search}{-}\mathrm{P}^{\mathrm{promise}-(\mathrm{IP}\cap\mathrm{coIP})} = \mathrm{psdIP}.$$ ◀

8 Discussion and Open Problems

Pseudo-determinism and TFNP: The class of total search problems solvable by pseudo-deterministic NP algorithms is a very natural subset of TFNP, the set of all total NP search problems. It is interesting to understand how the set of total psdNP problems fits in TFNP. For example, it is not known whether TFNP = psdNP. It would be interesting either to show that every problem in TFNP has a pseudo-deterministic NP algorithm, or to show that under plausible assumptions there is a problem in TFNP which does not have a pseudo-deterministic NP algorithm.

Similarly, it is interesting to understand the relationship of psdNP to other subclasses of TFNP. For example, one can ask whether every problem in PPAD has a pseudo-deterministic NP algorithm (i.e., given a game, does there exists a pseudo-deterministic NP or AM algorithm which outputs a Nash Equilibrium), or whether under plausible assumptions this is not the case. Similar questions can be asked for CLS, PPP, and so on.

Pseudo-determinism in Lattice problems: There are several problems in the context of lattices which have NP (and often also NP$\cap$coNP) algorithms [1]. Notable examples include gap-SVP and gap-CVP, for certain gap sizes. It would be interesting to show pseudo-deterministic interactive proofs for those problems. In other words, one could ask: does there exists an AM protocol for gap-SVP so that when a short vector exists, the *same* short vector is output every time. Perhaps more interesting would be to show, under plausible cryptographic assumptions, that certain such problems *do not* have psdAM protocols.

Pseudo-determinism and Number Theoretic Problems: The problem of generating primes (given a number n, output a prime greater than n), and the problem of finding primitive roots (given a prime p, find a primitive root mod p) have efficient randomized algorithms, and have been studied in the context of pseudo-determinism [13, 6, 22], though no polynomial time pseudo-deterministic algorithms have been found. It is interesting to ask whether these problems have polynomial time psdAM protocols.

The Relationship between psdAM **and** search−BPP**:** One of the main open problems in pseudo-determinism is to determine whether every problem in search−BPP also has a polynomial time pseudo-deterministic algorithm. This remains unsolved. As a step in that direction (and as an interesting problem on its own), it is interesting to determine whether search−BPP $\subseteq$ psdAM. In this paper, we proved a partial result in this direction, namely that search−BPP $\subseteq i.o.\text{psdMA}_{\text{SUBEXP}}$.

Zero Knowledge Proofs of Uniqueness: The definition of pseudo-deterministic interactive proofs can be extended to the context of Zero Knowledge. In other words, the verifier gets no information other than the answer, and knowing that it is the unique/canonical answer. It is interesting to examine this notion and understand its relationship to psdAM.

The Power of the Prover in pseudo-deterministic interactive proofs: Consider a search problem which can be solved in IP where the prover, instead of being all-powerful, is computationally limited. We know that such a problem can be solved in psdIP if the prover has unlimited computational power (in fact, one can show it is enough for the prover to be in PSPACE). In general, if the prover can be computationally limited for some IP protocol, can it also be computationally limited for a psdIP protocol for the same problem? It is also interesting in general to compare the power needed for the psdIP protocol compared to the power needed to solve the search problem non-pseudo-deterministically. Similar questions can be asked in the context of AM.

The Power of the Prover in pseudo-deterministic private vs public coins proofs: In our psdAM protocol for Graph Isomorphism, the verifier uses private coins, and the prover is weak (it can be simulated by a polynomial time machine with an oracle for graph isomorphism). If using public coins, what power would the prover need? In general, it is interesting to compare the power needed by the prover when using private coins vs public coins in psdAM and psdIP protocols.

Pseudo-deterministic interactive proofs for setting cryptographic global system parameters: Suppose an authority must come up with global parameters for a cryptographic protocol (for instance, a prime p and a primitive root g of p, which would be needed for a Diffie-Hellman key exchange). It may be important that other parties in the protocol know that the authority did not come up with these parameters because he happens to have a trapdoor to them. If the authority proves to the other parties that the parameters chosen are canonical, the other parties now know that the authority did not just pick these parameters because of a trapdoor (instead, the authority had to pick those parameters, since those are the canonical ones). It would be interesting to come up with a specific example of a protocol along with global parameters for which there is a pseudo-deterministic interactive proof showing the parameters are unique.

References

1 Dorit Aharonov and Oded Regev. Lattice problems in NP ∩ coNP. *Journal of the ACM (JACM)*, 52(5):749–765, 2005.

2 László Babai. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 421–429. ACM, 1985.

3 László Babai and Eugene M Luks. Canonical labeling of graphs. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 171–183. ACM, 1983.

4 J. Cai, V. Chakaravarthy, L. Hemaspaandra, and M. Ogihara. Competing provers yield improved Karp–Lipton collapse results. *Information and Computation*, 198(1):1–23, 2005.

5 John J Cannon. Construction of defining relators for finite groups. *Discrete Mathematics*, 5(2):105–129, 1973.

6 Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 18, page 136, 2011.

7 Oded Goldreich. *Modern cryptography, probabilistic proofs and pseudorandomness*, volume 17. Springer Science & Business Media, 1998.

8 Oded Goldreich, Shafi Goldwasser, and Dana Ron. On the possibilities and limitations of pseudodeterministic algorithms. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 127–138. ACM, 2013.

9 Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.

10 Shafi Goldwasser and Ofer Grossman. Perfect bipartite matching in pseudo-deterministic RNC. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 22, page 208, 2015.

11 Shafi Goldwasser and Michael Sipser. Private coins versus public coins in interactive proof systems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 59–68. ACM, 1986.

12 Joachim Grollmann and Alan L Selman. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335, 1988.

13 Ofer Grossman. Finding primitive roots pseudo-deterministically. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 22, page 207, 2015.

14 E. Hemaspaandra, L. Hemaspaandra, and C. Menton. Search versus decision for election manipulation problems. In *Proceedings of the 30th Annual Symposium on Theoretical Aspects of Computer Science*, pages 377–388. Leibniz International Proceedings in Informatics (LIPIcs), 2013.

15 L. Hemaspaandra and D. Narváez. The opacity of backbones. In *AAAI-2017*, pages 3900–3906. AAAI Press, 2017.

16 Lane A Hemaspaandra, Ashish V Naik, Mitsunori Ogihara, and Alan L Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM Journal on Computing*, 25(4):697–708, 1996.

17 Dhiraj Holden. A note on unconditional subexponential-time pseudo-deterministic algorithms for BPP search problems. *arXiv preprint arXiv:1707.05808*, 2017.

18 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on computing*, 17(5):935–938, 1988.

19 Rudolf Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–136, 1979.

20 Peter Bro Miltersen, N Variyam Vinodchandran, and Osamu Watanabe. Super-polynomial versus half-exponential circuit size in the exponential hierarchy. In *International Computing and Combinatorics Conference*, pages 210–220. Springer, 1999.

21 Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of computer and System Sciences*, 49(2):149–167, 1994.

22 Igor C Oliveira and Rahul Santhanam. Pseudodeterministic constructions in subexponential time. *arXiv preprint arXiv:1612.01817*, 2016.

23 Omer Reingold, Guy N Rothblum, and Ron D Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 49–62. ACM, 2016.

24 Adi Shamir. IP=PSPACE. *Journal of the ACM (JACM)*, 39(4):869–877, 1992.

25 Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

A Alternate Algorithm for Graph Isomorphism in pseudo-deterministic AM

In this section, we present another psdAM algorithm for Graph Isomorphism, this one more group theoretic (as opposed to the more combinatorial approach of the algorithm in Section 3). The method we use to do this involves finding the lexicographically first isomorphism using group theory. In particular, the verifier will obtain the automorphism group of one of the graphs from the prover and verify that it is indeed the automorphism group, and then the verifier will convert an isomorphism obtained from the prover into the lexicographically first isomorphism between the two graphs. We will define the group-theoretic terms used below.

▶ **Definition 20** (Automorphism Group)**.** The *automorphism group* $Aut(G)$ of a graph is the set of permutations $\phi : G \to G$ such that for every $u, v \in V(G)$, $(u, v) \in E(G) \iff (\phi(u), \phi(v)) \in E(G)$ (i.e., ϕ is an automorphism of G).

▶ **Definition 21** (Stabilize)**.** Given a set S and elements $\alpha_1, \alpha_2, ..., \alpha_i \in S$, we say that a permutation $\phi : S \to S$ *stabilizes* $\{\alpha_1, \alpha_2, ..., \alpha_k\}$ iff $\phi(\alpha_i) = \alpha_i$ for $i \in \{1, ..., k\}$. We also say that a group G *stabilizes* $\{\alpha_1, \alpha_2, ..., \alpha_k\}$ when every $\phi \in G$ stabilizes $\{\alpha_1, \alpha_2, ..., \alpha_k\}$.

▶ **Definition 22** (Stabilizer)**.** The *stabilizer* of an element s in S for a group G acting on S is the set of elements of G that stabilize s.

▶ **Lemma 23.** *Suppose that we are given a tuple (G_1, G_2, H, ϕ) where G_1 and G_2 are graphs, $H = Aut(G_1)$ is represented as a set of generators, and ϕ an isomorphism between G_1 and G_2. Then, in polynomial time, we can compute a unique isomorphism ϕ^* from G_1 to G_2 independent of the choice of ϕ and the representation of H.*

Proof. We use the algorithm given in [5] to compute a canonical coset representative, observing that the set of isomorphisms between G_1 and G_2 is a coset of the automorphism group of G_1. Let $\alpha_1, ..., \alpha_t$ be a basis of H, i.e., a set such that any $h \in H$ fixing $\alpha_1, ..., \alpha_t$ is the identity. Let H_i be the subgroup of H that stabilizes $\alpha_1, ..., \alpha_{i-1}$. Now, let U_i be a set of coset representatives of H_{i+1} in H_i. Given the generators of H_i, we can calculate U_i, and by Schreier's theorem we can calculate the generators for H_{i+1}. In this fashion, we can get generators and coset representatives for all the H_i. To produce ϕ^*, we do the following.

```
Find-First-Isomorphism
1  $\phi^* = \phi$
2  For $i = 1, ..., t$
3      Let $P_i = \{\phi^* u | u \in U_i\}$.
4      Set $\phi^* = \arg\min_{\phi \in P_i}(\phi(\alpha_i))$.
```

To see that this produces a unique isomorphism that does not depend on ϕ, observe that $\phi^*(\alpha_1)$ is the minimum possible value of $\phi(\alpha_1)$ over all isomorphisms of G_1 to G_2 as U_1 is a set of coset representatives for the stabilizer of α_1 over H. Also, if $\phi^*(\alpha_i)$ is fixed for $i \in \{1, ..., k\}$, then $\phi^*(\alpha_{k+1})$ is the minimum possible value of $\phi(\alpha_{k+1})$ over all isomorphisms which take α_1 to $\phi^*(\alpha_1)$, α_2 to $\phi^*(\alpha_2)$,..., and α_k to $\phi^*(\alpha_k)$, as U_{i+1} stabilizes $\alpha_1, ..., \alpha_k$, so everything in P_{i+1} takes α_1 to $\phi^*(\alpha_1)$, α_2 to $\phi^*(\alpha_2)$,..., and α_k to $\phi^*(\alpha_k)$. This implies that ϕ^* does not depend on ϕ and is unique. ◀

Given this result, this means that it suffices to show a protocol that lets the verifier obtain a set of generators for the automorphism group of G_1 and an isomorphism that are correct with high probability, as by the above lemma this can be used to obtain a unique isomorphism between G_1 and G_2 independent of the isomorphism or the generators.

▶ **Theorem 24.** *There exists an interactive protocol for graph isomorphism such that with high probability, the isomorphism that is output by the verifier is unique, where in the case of a cheating prover the verifier fails instead of outputting a non-unique isomorphism. In other words, finding an isomorphism between graphs can be done in* psdAM.

Proof. From Lemma 23, it suffices to show an interactive protocol that computes the automorphism group of a graph in a verifiable fashion. [19] reduces the problem of computing the generators of the automorphism group to the problem of finding isomorphisms. Using this reduction, we can make a constant-round interactive protocol to determine the automorphism group by finding the isomorphisms in parallel. The reason we can do this in parallel is that [19] implies that there are $O(n^4)$ different pairs of graphs to check and for each pair of graphs we either run the graph isomorphism protocol or the graph non-isomorphism protocol. In the case of the graph isomorphism protocol, the verifier need only accept with an isomorphism in hand; for graph non-isomorphism, the messages sent to the prover are indistinguishable between the two graphs when they are isomorphic, so since the graphs and permutations are chosen independently, there is no way for the prover to correlate their answers to gain a higher acceptance probability for isomorphic graphs. Thus this means that the verifier can determine the automorphism group of a graph and verify that it is indeed the entire automorphism group. Using Lemma 23 we then see that the prover just has to give the verifier an isomorphism, and verifier can compute a unique isomorphism using the automorphism group. ◀