Electronic Colloquium on Computational Complexity, Report No. 135 (2019)

Doubly-Efficient Pseudo-Deterministic Proofs

Michel Goemans[1], Shafi Goldwasser[2], and Dhiraj Holden[3]

[1]Department of Mathematics, MIT goemans@math.mit.edu
[2]Simons Institute for the Theory of Computing, UC Berkeley shafi.goldwasser@gmail.com
[3]Department of Electrical Engineering and Computer Science, MIT dholden@mit.edu

October 2, 2019

Abstract

In [20] Goldwasser, Grossman and Holden introduced pseudo-deterministic interactive proofs for search problems where a powerful prover can convince a probabilistic polynomial time verifier that a solution to a search problem is canonical. They studied search problems for which polynomial time algorithms are not known and for which many solutions are possible. They showed that whereas there exists a constant round pseudo deterministic proof for graph isomorphism where the canonical solution is the lexicographically smallest isomorphism, the existence of pseudo-deterministic interactive proofs for NP-hard problems would imply the collapse of the polynomial time hierarchy.

In this paper, we turn our attention to studying *doubly-efficient pseudo-deterministic proofs* for polynomial time search problems: pseudo-deterministic proofs with the extra requirement that the prover runtime is polynomial and the verifier runtime to *verify* that a solution is canonical is significantly lower than the complexity of *finding* any solution, canonical or otherwise. Naturally this question is particularly interesting for search problems for which a lower bound on its worst case complexity is known or has been widely conjectured.

We show doubly-efficient pseudo-deterministic algorithms for a host of natural problems whose complexity has long been conjectured. In particular,

- We show a doubly efficient pseudo-deterministic proof for **linear programming** where the canonical solution which the prover will provide is the lexicographically greatest optimal solution for the LP. To this end, we show how through perturbing the linear program and strong duality this solution can be both computed efficiently by the prover, and verified by the verifier. The time of the verifier is $O(d^2)$ for a linear program with integer data and at most d variables and constraints, whereas the time to solve such linear program is $\tilde{O}(d^\omega)$ by randomized algorithms [11] for ω the exponent for fast matrix multiplication .
- We show a doubly efficient pseudo-deterministic proof for **3-SUM** and problems reducible to 3-SUM where the prover is a $O(n^2)$ time algorithm and the verifier takes time $\tilde{O}(n^{1.5})$.
- We show a doubly-efficient pseudo-deterministic proof for the **hitting set problem** where the verifier runs in time $\tilde{O}(m)$ and the prover runs in time $\tilde{O}(m^2)$ where $m = \sum_{S\in\mathcal{S}} |S| + \sum_{T\in\mathcal{T}} |T|$ for inputs collections of sets $\mathcal{S}, \mathcal{T}$.
- We show a doubly-efficient pseudo-deterministic proof for the **Zero Weight Triangle problem** where the verifier runs in time $\tilde{O}(n^{2+\omega/3})$ and the prover runs in randomized time $\tilde{O}(n^3)$. The Zero Weight Triangle problem is equivalent to the **All-Pairs Shortest Path problem**, a well-studied problem that is the foundation of many hardness results in graph algorithms [39, 38], under sub-cubic reductions.

ISSN 1433-8092

1 Introduction

Pseudo-deterministic algorithms, introduced by Gat and Goldwasser [17], are probabilistic (polynomial-time) algorithms for search problems that, with high probability, *find* a unique output for each input except with negligible error probability. Such output for input x is referred to as the "canonical" output for x. Algorithms that satisfy the aforementioned condition are of importance whenever uniqueness or "reproducibility" of the answer is important. This is of particular relevance in a distributed or parallel setting when an algorithm is executed by multiple parties for whom it is challenging (for reasons of trust or efficiency requirement) to *agree* on a common sequence of unbiased random coins.

More recently, Goldwasser, Grossman and Holden [20] extended the study to pseudo-deterministic interactive proofs for search problems, denoted psdIP. The new goal was to *prove* to a probabilistic polynomial time verifier that a solution to a search problem is canonical. The motivation was to address those search problems for which polynomial time algorithms are not known and for which many solutions are possible, such as for graph isomorphism. In this case the *search problem* is to find an isomorphism between two graphs if one exists and an example of a *canonical solution* would be the lexicographically smallest isomorphism. 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 a malicious prover should not be able to convince the verifier to accept any solution other than the unique canonical one and that the interaction should be constant round. If unbounded number of rounds are allowed, the $IP = PSPACE$ characterization implies that psdIP $= IP$.

In this paper, we turn our attention to studying *doubly-efficient pseudo-deterministic proofs*. That is pseudo-deterministic proofs with the extra requirement that the prover is efficient as well. Our aim is to show doubly-efficient pseudo-deterministic proofs for polynomial time problems, where the prover runs in polynomial time in the complexity of the problem and the verifier can *verify* that a solution is canonical significantly more efficiently than solving the problem without the presence of the prover. We remark that in the doubly-efficient pseudo-deterministic proofs below, except for linear programming, the runtime of the prover is at most a constant times the runtime of the best known deterministic algorithm.

1.1 Our Results

A new notion: Doubly-efficient pseudo-deterministic interactive proofs

We define *doubly-efficient pseudo-deterministic interactive proofs* for a search problem R of complexity $T(n)$ (consisting of pairs $(instance, solution)$) with associated canonization function c as a pair of interacting algorithms: a probabilistic polynomial time prover which runs in time $poly(T(n))$ and a probabilistic verifier which runs in time $o(T(n))$ which on a common input instance x engage in constant number of rounds of interaction at the end of which with high probability the verifier outputs a canonical solution $y = c(x)$ if any solution exists and otherwise rejects x. Analogously to the case of completeness in interactive proofs for languages, we require that for every input x, there exists an honest prover which can send the correct solution $c(x)$ to the verifier when one exists. Analogously to the case of soundness, no dishonest prover can cause the verifier to output a solution other than $c(x)$ (the canonical one) (except with very low probability).

A few remarks are in order.

- Naturally this question is particularly interesting for search problems for which a lower bound on its worst case complexity $T(n)$ is known or has been widely conjectured. This will drive our choice of problems for which we show doubly efficient pseudo-deterministic proofs.
- Doubly-efficient pseudo-deterministic proofs for search problems R with associated canonization function c are closely related to **computation delegation** of computing $c(x)$ on input x. The delegation problem was posed by Goldwasser, Kalai, and Rothblum [21] and

become known under the name doubly-efficient interactive proof systems. The difference in the requirements is that [21] allow the prover to be any polynomial time algorithm and the verifier to run in linear (up to log factors) time and addresses deterministic computations. Doubly-efficient interactive proofs have been shown by [21] for log-space uniform sets in NC (or, more generally, to inputs that are acceptable by log-space uniform bounded-depth circuits, where the number of rounds in the proof system is linearly related to the depth of the circuit). Reingold, Rothblum and Rothblum [25] showed that any set decidable in polynomial-time by an algorithm of space complexity $s(n) \leq n^{0.499}$, has a constant-round interactive proof system in which the prover runs in polynomial time and the verifier runs in time $\tilde{O}(n)$. Finally Goldreich and Rothblum [19] show direct constructions of doubly-efficient interactive proof systems for problems in P that are believed to have relatively high complexity such as t-CLIQUE and t-SUM.

We remark that works on proof systems and delegation did not stay within the realm of theory alone. Rather, they became the theoretical basis for several system implementations of a delegation system as they offered reasonably efficiently realizable protocols. Indeed, there is a flourishing literature surrounding the refinement and implementation of these theoretical protocols [2, 3, 26, 4, 5, 7, 10, 12, 13, 15, 22, 23, 27, 29, 28, 30, 31, 34, 35] (see [36] for a survey). We hope that our proposed study of doubly-efficient pseudo-deterministic proofs can similarly impact practice (and beyond).

- The setting of doubly-efficient interactive proofs naturally models a cryptographic setting where users wish to have access to common cryptographic system-wide keys or parameters, such as a pair (g, p) for Z_p with prime p and generator g for a given input length n. A central authority (with additional computational power) can of course choose the common system-wide parameter and broadcast it to all, but then who is to say that the central party did not chose its randomness in a way that would force an output for which the trusted center knew some "trapdoor" information which would enable it to break the underlying cryptographic security? Viewing the generation of a cryptographic key as a solution to a search problem R per security parameter, a doubly-efficient pseudo-deterministic proof for R would ensure that the prover had no choice in which parameter to broadcast as he could prove that his solution is canonical.

Doubly-efficient pseudo-deterministic algorithms for linear programming and fine-grained complexity problems

Linear Programming: We show a doubly-efficient pseudo-deterministic proof for the linear programming problem. Verifying an optimal solution to a linear programming problem can be done thanks to *strong duality*: there exists a solution to the dual problem with the same value as the solution to the primal problem. We show that a special optimal solution, namely the lexicographically greatest solution, can be efficiently obtained by the prover, and that the prover can convince the verifier that the LP solution it gives to the verifier is indeed the lexicographically greatest solution; this is done through perturbing the linear program and strong duality. More concretely, every linear program (say, where the objective is to maximize) has a corresponding dual linear program, a minimization problem, with the property that (i) (weak duality) any feasible solution to the dual provides an upper bound on the optimal primal value and (ii) (strong duality) there exists an optimal solution to the dual with the same value as the primal optimal solution. Furthermore, there exist compact polynomial-sized solutions to the primal and dual linear programs. Therefore such a polynomial-sized feasible solution to the dual with an equal value as a primal solution provides a compact certificate for the optimality of this primal solution.

The currently best known time to solve a linear program with integer data and at most d variables and constraints is $\tilde{O}(d^{\omega})$ randomized [11] where ω corresponds to the exponent for fast matrix multiplication which is currently at ≈ 2.37 and $\tilde{O}()$ hides polylog factors including a $\log(1/\delta)$ factor to account for the accuracy δ in solving the linear program. The time of the verifier to verify a pair of primal and dual optimal solution is only $O(d^2)$ as this only requires matrix-vector multiplication.

Problems studied in fine-grained complexity: We next show doubly-efficient pseudo-deterministic proofs for several fine-grained complexity problems where the verifier significantly beats the conjectured time. The challenge is to find a proof where the prover's running time does not change too much from the running time of the deterministic algorithm. In the case of two of our problems, making the running time close to the running time of the deterministic algorithm requires the prover to run in a randomized fashion.

The **3-SUM problem** has an easy $O(n^2)$ time algorithm which can be improved by polylogarithmic factors. It is an outstanding open question whether there is an algorithm that significantly improves $O(n^2)$. Finding such an algorithm would yield algorithms for a host of other problems in computational geometry [16, 14]. Here, we show a doubly-efficient pseudo-deterministic proofs that outputs the lexicographically first such triple of elements where the verifier takes time $\tilde{O}(n^{1.5})$. We crucially use the fact that [8] gives a nondeterministic proof that there is no triple of elements that sum to 0 where the verifier takes time $\tilde{O}(n^{1.5})$.

The **hitting set problem** is the problem of finding a set in a collection of sets that intersects every set in a different collection of sets. We show a pseudo-deterministic proof for the hitting set problem where the verifier runs in time $\tilde{O}(m)$ and the prover runs in time $\tilde{O}(m^2)$ where $m = \sum_{S \in \mathcal{S}} |S| + \sum_{T \in \mathcal{T}} |T|$ for inputs $\mathcal{S}, \mathcal{T}$ collections of sets. This problem has been conjectured to take $m^{2-o(1)}$ time [33].

The **All-Pairs Shortest Path** problem is a well-studied problem that is the foundation of many hardness results in graph algorithms [39, 38]. In particular, the **Zero Weight Triangle** problem is equivalent to the All-Pairs Shortest Path problem under subcubic reductions. We show a doubly efficient pseudo-deterministic proof for the Zero Weight Triangle problem where the verifier runs in time $\tilde{O}(n^{2+\omega/3})$ and the prover runs in randomized time $\tilde{O}(n^3)$.

Techniques

All our results take on the following flavor: For a search problem R, the pseudo-deterministic algorithm, given x, finds the lexicographically first y such that $R(x, y)$. To do this, it asks whether there exists y' such that $(x, 0y') \in R$, y' such that $(x, 1y') \in R$, etc. and finds the first y such that $R(x, y)$ recursively. The notion of "lexicographically first" can be easily generalized to allow other orderings and other encodings of the input. This suggests that more generally doubly-efficient pseudo-deterministic proofs for search are the ones where there is a doubly-efficient proof of existence and a doubly-efficient proof of nonexistence of solutions to said search problem. A more general theorem (Lemma 3.3) follows under general conditions.

2 Preliminaries

In this section we will introduce concepts needed to give pseudo-deterministic proofs that improve on the best known deterministic algorithms for problems studied in fine-grained complexity.

Definition 2.1 (Search Problem)**.** A *search problem* is a relation R consisting of pairs (x, y) and we define L_R to be the set of x such that $\exists y(x, y) \in R$.

The goal of an algorithm solving a search problem is to find a y such that $(x, y) \in R$. The focus of pseudo-determinism is to give algorithms for search problems that find canonical solutions; a pseudo-deterministic algorithm will output the same solution to a search problem with high probability over its randomness. [20] extended the notion of pseudo-determinism to interactive proofs and brought the concept of NP search problems with unique answers under the umbrella of pseudo-determinism. We will refer to this work's definition of a pseudo-deterministic proof. The pseudo-deterministic proofs in our setting will always either output the unique solution or $\perp$.

Definition 2.2 (Pseudo-deterministic proof [20])**.** A search problem R is in *pseudo-deterministic* IP (often denoted psdIP) if there exists a function s where all $x \in L_R$ satisfy $(x, s(x)) \in R$, and 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', $\Pr_r[(P', V)(x, r) \neq \perp] \leq \frac{1}{3}$.

This is analogous to the definition of pseudo-deterministic NP, except we allow the prover and verifier to interact. In the setting we consider, the prover and verifier both run in polynomial time, with the prover given more time than the verifier. Our goal is to construct pseudo-deterministic proofs for problems such that the verifier runs in time faster than the best known deterministic algorithm for the problem.

3 Doubly-efficient pseudo-deterministic proofs

We want to extend the concept of pseudo-deterministic proofs to the setting where the prover also runs in polynomial time, and we want to extend the concept of doubly-efficient interactive proofs to the setting where the verifier outputs a unique solution. Both of these tasks are accomplished by introducing *doubly-efficient pseudo-deterministic proofs* : proofs where both the verifier and prover run in polynomial time, the verifier running in time asymptotically faster, and where the verifier will output a unique solution given an input.

Definition 3.1. A $(t_1(n), t_2(n))$ pseudo-deterministic proof is a pseudo-deterministic proof where the verifier V runs in (probabilistic) time $t_1(n)$ and the prover P runs in (probabilistic) time $t_2(n)$.

Ideally, we want the prover to run in time almost equal to the deterministic running time of the problem, as this means the total work is not much more than the work of solving this problem deterministically. However, we say that pseudo-deterministic proof is *non-trivial* as long as the verifier runs faster than the deterministic running time of the problem. To demonstrate the concept, we will consider the pseudo-deterministic proof for graph isomorphism. The prover from [20] only needs the power to compute n^2 instances of graph isomorphism. We know from [1] that graph isomorphism is in quasi-polynomial time. Thus, the result of [20] about graph isomorphism can be restated as:

Corollary 3.2. *Graph Isomorphism has a $(poly(n), quasipoly(n))$ pseudo-deterministic proof.*

A large class of pseudo-deterministic algorithms have the following format: for a search problem R, the pseudo-deterministic algorithm, given x, finds the lexicographically first y such that $R(x, y)$. To do this, it asks whether there exists y' such that $(x, 0y') \in R$, y' such that $(x, 1y') \in R$, etc. and finds the first y such that $R(x, y)$ recursively. For instance, [17] gives a pseudo-deterministic algorithm for testing if a polynomial is non-zero by finding the lexicographically first non-zero solution. Given $p(x_1, ..., x_n)$, the algorithm tests if $p(0, ..., x_n)$ is zero everywhere. If it is not zero everywhere, then the algorithm checks if $p(0, 0, ..., x_n)$ is zero everywhere, and otherwise the algorithm checks if $p(1, ..., x_n)$ is zero everywhere. This continues recursively until the algorithm finds the first element that is non-zero or rejects. Also, [20] provides a pseudo-deterministic proof for graph isomorphism where the verifier outputs the lexicographically first isomorphism by going recursively. The algorithm starts by figuring out where the first vertex is mapped in the lexicographically first isomorphism by looping through the vertices, then where the second vertex is mapped, and so on until the lexicographically first isomorphism has been found.

We will use a structure similar to this to define doubly-efficient pseudo-deterministic proofs for a large class of problems studied within the fine grained complexity literature.

Lemma 3.3. *Suppose we have a search problem $R(x, y)$ such that $|y| = poly(x)$, finding the lexicographically first y given x such that $R(x, y)$ takes time $t_1(n)$, computing $R(x, y)$ takes time $t_2(n)$, and y can be written as $y_1...y_k$ such that the following holds:*

- *Given $x, y_1, ..., y_i$, the problem $\exists z_i < y_i \exists y_{i+1}, ..., y_k R(x, y_1, ..., y_{i-1}, z_i, y_{i+1}...y_k)$ can be solved in co-nondeterministic time $t_3(n)$ where the prover runs in time $t_4(n)$.*

Then there exists a $(t_2(n) + k * t_3(n), t_1(n) + k * t_4(n))$ pseudo-deterministic proof that outputs the lexicographically first y such that $R(x, y)$.

Proof. Our algorithm proceeds in two stages: in the first stage, the prover gives y, taking time $t_1(n)$ to find the lexicographically first y such that $R(x, y)$, and the verifier checks whether $R(x, y)$ and outputs $\perp$ otherwise; this takes time $t_2(n)$. In the next stage we prove that y is the lexicographically first such y; that is, for all $z <_{lex} y$, $\neg R(x, z)$. To do so, we only need to check that there is no i such that $\exists z_i < y_i \exists z_{i+1}, ..., z_k$ such that $R(y_1, y_2, ..., y_{i-1}, z_i, z_{i+1}, ..., z_k)$ for $1 \leq i \leq k$. Since we have to do this k times, the total time of this stage is $k * (t_2(n))$ for the verifier and $k * t_4(n)$ for the prover. Our algorithm clearly outputs the lexicographically first y such that $R(x, y)$, and a cheating prover cannot make the verifier output a different y. □

Now that we have shown a general framework for constructing pseudo-deterministic proofs, we will proceed to show a number of problems for which there exist pseudo-deterministic proofs where the verifier runs in time faster than the best known deterministic algorithms. In addition, there is evidence suggesting that the best known deterministic algorithms are nearly optimal; in particular, there is evidence against being able to turn these pseudo-deterministic proofs into deterministic algorithms where the running time of the deterministic algorithm is the same as the running time of the verifier for the pseudo-deterministic proof.

4 Linear programming

In [20], we use the fact that graph non-isomorphism has an AM proof to give a pseudo-deterministic AM proof for graph isomorphism. Here we show a pseudo-deterministic proof for linear programming. Linear programming is the class of optimization problems with linear constraints and a linear objective function. We exploit the fact that linear programming admits a good characterization, a compact way of certifying the optimality of a solution. Indeed every linear program (say, where the objective is to maximize) has a corresponding dual linear program, a minimization problem, with the property that (i) (weak duality) any feasible solution to the dual provides an upper bound on the optimal primal value and (ii) (strong duality) there exists an optimal solution to the dual with the same value as the primal optimal solution. Furthermore, there exist compact polynomial-sized solutions to the primal and dual linear programs. Therefore such a polynomial-sized feasible solution to the dual with an equal value as a primal solution provides a compact certificate for the optimality of this primal solution.

In order to be able to turn this into a pseudo-deterministic proof, we need the prover to identify a special, unique optimal solution (as there could be a continuum of primal optimal solutions), and provide a way for the verifier to efficiently verify it. As special solution, we use the *lexicographically greatest* optimal solution to the primal. Among all optimal solutions, the lexicographically greatest first maximizes x_1, then x_2, and so on; see below for a precise definition. To verify it, one option would be to provide dual optimal solutions to a squence of dual linear programs corresponding to the definition of lexicographically greatest maximal solution. A better (more efficient) way, which we describe in this section, is to show that we can perturb the objective function of the primal linear program in such a way that there is a unique optimal solution and that this solution is the unique lexicographically greatest optimal solution for the unperturbed linear program.

We start with basic notation and linear programming fundamentals.

Definition 4.1. A linear program is the problem $\max\{\mathbf{c}^\top\mathbf{x}\}$ subject to the constraints $A\mathbf{x} \le \mathbf{b}$ and $\mathbf{x} \ge \mathbf{0}$. Its dual is the linear program $\min\{\mathbf{b}^\top\mathbf{y}\}$ subject to the constraints $A^\top\mathbf{y} \ge \mathbf{c}$ and $\mathbf{y} \ge \mathbf{0}$.

Theorem 4.2. *(Weak duality) If $\mathbf{x}, \mathbf{y}$ are feasible solutions to a linear program given by $\max\{\mathbf{c}^\top\mathbf{x}\}$ subject to $A\mathbf{x} \le \mathbf{b}$ and $\mathbf{x} \ge \mathbf{0}$ and its dual respectively, then $\mathbf{c}^\top\mathbf{x} \le \mathbf{b}^\top\mathbf{y}$. (Strong duality) Furthermore $\mathbf{x}, \mathbf{y}$ are optimal solutions if and only if $\mathbf{c}^\top\mathbf{x} = \mathbf{b}^\top\mathbf{y}$.*

Furthermore, there exist optimal solutions of polynomial size, since any extreme point (which cannot be expressed as a strict convex combination of feasible points) has this property.

Theorem 4.3 ([18])**.** *Let P be the linear program given by $\max \mathbf{c}^\top\mathbf{x}$ subject to $A\mathbf{x} \le \mathbf{b}$, where all inputs are integers and A is an $m \times n$ matrix. Define $L = m + n + \log(\max_{A'} |det(A')|) + \log(\max_i |b_i|) + \log(\max_j |c_j|)$, where A' range over all square submatrices of A. Then any extreme point $\mathbf{x}$ of P is of the form $x_i = \frac{p_i}{q}$ where q and p_i's are integers satisfying $1 \le q < 2^L$ and $0 \le p_i < 2^L$ for all i.*

This quantity L is often used when referring to efficiency of linear programming algorithms, and can be seen (see [18]) to be polynomially related to the binary encoding of all the input data.

Definition 4.4. The *lexicographically greatest* optimal solution $\mathbf{x}^*$ to a linear program $\max\{\mathbf{c}^\top\mathbf{x}\}$ subject to $A\mathbf{x} \le \mathbf{b}$ and $\mathbf{x} \ge \mathbf{0}$ is the solution that satisfies (i) feasibility: $A\mathbf{x}^* \le \mathbf{b}$ and $\mathbf{x}^* \ge \mathbf{0}$, (ii) optimality: $\mathbf{c}^\top\mathbf{x}^* = \max_{A\mathbf{x}\le\mathbf{b},\mathbf{x}\ge\mathbf{0}}\{\mathbf{c}^\top\mathbf{x}\}$, and (iii) for every $\mathbf{x} \in \arg\max_{A\mathbf{x}\le\mathbf{b},\mathbf{x}\ge\mathbf{0}}\{\mathbf{c}^\top\mathbf{x}\}$, either $\mathbf{x} = \mathbf{x}^*$ or there exists $i \le n$ with $x_i < x_i^*$ and $x_j = x_j^*$ for $j < i$.

Now that we have defined the necessary terminology, we can proceed to proving that linear programming has a pseudo-deterministic interactive proof. To do so, we perturb our linear program so that the only optimal solution to the new linear program is the lexicographically greatest solution to the original program, and then use the dual linear program to prove that the solution given to the verifier is optimal.

Theorem 4.5. *Let P be the linear program given by $\max \mathbf{c}^\top\mathbf{x}$ subject to $A\mathbf{x} \le \mathbf{b}$, with L defined as above. Then, the linear program P' given by $\max \mathbf{c}^\top\mathbf{x} + \epsilon x_1 + \epsilon^2 x_2 + \ldots + \epsilon^n x_n$, where $\epsilon = 2^{-3L-2}$, has a unique solution which is the lexicographically greatest solution of P.*

Proof. First consider the unperturbed linear program P, and two extreme point solutions $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, with corresponding denominators q_1 and q_2 respectively (see Theorem 4.3).

If $\mathbf{c}^\top\mathbf{x}^{(1)} > \mathbf{c}^\top\mathbf{x}^{(2)}$ then $\mathbf{c}^\top\mathbf{x}^{(1)} - \mathbf{c}^\top\mathbf{x}^{(2)} \ge \frac{1}{q_1 q_2} > 2^{-2L}$. Let $\mathbf{c}'$ be the perturbed $\mathbf{c}$ (by adding the vector $(\epsilon, \epsilon^2, \cdots, \epsilon^n)$). Then

$$\mathbf{c}'^\top\mathbf{x}^{(1)} - \mathbf{c}'^\top\mathbf{x}^{(2)} > 2^{-2L} + \sum_{i=1}^{n} \epsilon^i (x_i^{(1)} - x_i^{(2)}) > 2^{-2L} - 2^L \sum_{i=1}^{n} \epsilon^i > 2^{-2L} - 2^L \epsilon/(1-\epsilon) > 0,$$

given our choice of ϵ. This shows that, after perturbation, we still have that $\mathbf{x}^{(1)}$ has a greater objective value than $\mathbf{x}^{(2)}$.

Suppose, on the other hand, that $\mathbf{c}^\top\mathbf{x}^{(1)} = \mathbf{c}^\top\mathbf{x}^{(2)}$ and that $\mathbf{x}^{(1)}$ is lexicographically greater than $\mathbf{x}^{(2)}$, i.e. that $x_i^1 > x_i^2$ while $x_j^1 = x_j^2$ for $j < i$. Then

$$\mathbf{c}'^\top\mathbf{x}^{(1)} - \mathbf{c}'^\top\mathbf{x}^{(2)} = \sum_{k=i}^{n} \epsilon^k (x_k^1 - x_k^2) \ge \epsilon^i \left(\frac{1}{q_1 q_2} - \sum_{\ell=1}^{n-i} \epsilon^\ell 2^L \right) > \epsilon^i \left(2^{-2L} - \frac{\epsilon}{1-\epsilon} 2^L \right) > 0,$$

showing that, after perturbation, the lexicographically greater solution $\mathbf{x}^{(1)}$ has greater (perturbed) objective function value. Together, this shows that the unique optimal solution to the perturbed problem is the lexicographically greatest solution to P. □

Observe that the parameter L' of the perturbed linear program increases polynomially to $O(nL)$, but the precision needed to solve the linear program approximately in order to be able to recover the unique extreme point solution is still $2^{-O(L)}$, as this represents a lower bound on the difference in value between any two extreme point solutions.

Theorem 4.6. *There exists a $(O(d^2 \log(1/\delta)), \tilde{O}(d^\omega \log(1/\delta)))$ pseudo-deterministic interactive proof for finding an optimal solution to a linear program P.*

Linear programs with at most d variables and constraints can be solved within an error of δ in time $\tilde{O}(d^{2.5} \log(1/\delta))$ deterministically [32], and in time $\tilde{O}(d^\omega \log(1/\delta))$ randomized [11] with ω (currently ~ 2.37) corresponds to the exponent for fast matrix multiplication. The notation $\tilde{O}$ hides polylog factors. The time to verify a pair of primal and dual optimal solution is only $O(d^2)$ (with a $\log(1/\delta)$ factor for bit complexity) as this only requires matrix vector multiplication. So, verification is currently more efficient than finding the solution.

Proof. By Theorem 4.5, we can perturb the objective function of P and obtain a linear program P' which has a unique solution, namely the lexicographically greatest solution of P. Let Q' be the dual linear program to P'. The prover sends over optimal solutions to P' and Q'. Then, the verifier checks to see whether the solutions are feasible and also whether the value of the solution to P' is equal to the value of the solution to Q'. If both of these conditions hold, the verifier outputs the solution to P', otherwise it outputs $\perp$. If the prover is honest, then clearly the verifier will output the solution to P'. A cheating prover cannot make the verifier output a different solution to P, as this would not correspond to an optimal solution of P' since it is unique. □

5 Problems studied in fine-grained complexity

5.1 3-SUM and problems reducible to 3-SUM

3-SUM is the problem to find 3 numbers that sum to 0, where the numbers are drawn from 3 lists. The 3-SUM problem has an easy $O(n^2)$ time algorithm and this can be improved by polylogarithmic factors [9]. It is an outstanding open question whether there is an algorithm that is much faster than $O(n^2)$, and finding such an algorithm would give faster algorithms for a host of other problems in computational geometry [16, 14]. We will show a pseudo-deterministic proof where the verifier runs in time $\tilde{O}(n^{1.5})$.

Definition 5.1. We say the 3-SUM problem is the problem of, given 3 lists $a_1, ..., a_n$, $b_1, ..., b_n$, $c_1, ..., c_n$, of $O(\log n)$ bit integers, finding a triple a_i, b_j, c_k such that $a_i + b_j + c_k = 0$.

In addition, [8] gives a nondeterministic proof that there is no triple of elements that sum to 0 where the verifier takes time $\tilde{O}(n^{1.5})$.

Theorem 5.2 ([8])**.** *3-SUM* $\in \text{coNTIME}(\tilde{O}(n^{1.5}))$.

Theorem 5.3. *There exists a non-deterministic proof for $\overline{\text{3-SUM}}$ where the verifier runs in time $\tilde{O}(n^{1.5})$ and the prover runs in randomized time $\tilde{O}(n^2)$.*

Proof. It takes a bit of work to show that the prover for this algorithm can run in randomized time $O(n^2)$. What the algorithm does is sends a prime p such that fewer than $\tilde{O}(n^{1.5})$ triples sum to 0 mod p, and all of the triples that add to 0 mod p. Since there are n^3 triples and each sum must be a product of at most $\log(n)$ primes, we get that there are $\tilde{O}(n^3)$ pairs (a_i, b_j, c_k, p) such that $a_i + b_j + c_k = 0 \pmod p$. Thus, in the first $n^{1.5}$ primes, over half the primes p will have $\tilde{O}(n^{1.5})$ triples that sum to 0 mod p by Markov's inequality. Thus if we sample a random prime in the first $n^{1.5}$ primes, we will get a good prime with high probability. Finding the sums equal to 0 mod p still takes time $\tilde{O}(n^2)$ deterministically. □

With this, we can construct a pseudo-deterministic proof for 3-SUM where the prover runs in time almost equal to the best known deterministic algorithm for 3-SUM.

Theorem 5.4. *3-SUM has a $(\tilde{O}(n^{1.5}), \tilde{O}(n^2))$ pseudo-deterministic proof.*

Proof. We split the answer y into $y_1 = i$, $y_2 = j$, and $y_3 = k$. We have a $\tilde{O}(n^{1.5})$ algorithm for proving that a list has no 3 integers which sum to 0. To check whether there is a 3-SUM with $z_i < y_i$, we can simply replace the first $i - 1$ lists with $y_1, ..., y_{i-1}$ respectively and take out all of the elements of the ith list after and including y_i, and then use a nondeterministic proof to show that there is no 3-SUM in these lists. It takes time $\tilde{O}(n^2)$ to find a 3-SUM and the prover takes randomized time $\tilde{O}(n^2)$ in the nondeterministic proof that there is no 3-SUM. Thus Lemma 3.3 implies that there is a pseudo-deterministic proof where the verifier runs in time $\tilde{O}(n^{1.5})$ and the prover runs in randomized time $\tilde{O}(n^2)$. □

Corollary 5.5. *Determining whether there are three collinear points in a set of points on the plane has a $(\tilde{O}(n^{1.5}), \tilde{O}(n^2))$ pseudo-deterministic proof.*

5.2 Hitting Set

The Hitting Set problem is, given two collections of sets, find a set in the first collection that intersects every set in the second collection. The Hitting Set problem is also conjectured to take $m^{2-o(1)}$ time [33]. Here we give a pseudo-deterministic proof in which the verifier runs in linear time.

Definition 5.6. The Hitting Set problem is, given two collections $\mathcal{S}, \mathcal{T}$ of sets, find a set S such that $S \cap T \neq \emptyset \forall T \in \mathcal{T}$.

Theorem 5.7 ([8])**.** *There is a nondeterministic proof where the verifier runs in time $O(m), m = \sum_{S \in \mathcal{S}} |S| + \sum_{T \in \mathcal{T}} |T|$, and the prover runs in time $O(m^2)$ for the Hitting Set problem and the complement of the Hitting Set problem.*

Theorem 5.8. *Hitting Set has a $(O(m), O(m^2))$ pseudo-deterministic proof.*

Proof. We can reduce the problem of showing there is no set S' that is a hitting set before S to Hitting Set by removing all of the sets after S including S and proving that there does not exist a hitting set for T. Then, by Lemma 3.3, this implies there exists a pseudo-deterministic proof for Hitting Set where the verifier runs in time $O(m)$ and the prover runs in time $O(m^2)$. □

5.3 Model checking of graph properties

A large number of different graph problems can be expressed as model checking of first-order properties as observed by [8]. For instance both the k-Dominating Set problem [24] and asking whether a graph has diameter 2 [6] can be written as model checking problems. [37] shows that given a first-order property of a graph with k quantifiers over vertices, checking whether the graph has this property can be done in time $\tilde{O}(n^{k-3+\omega})$. We extend the work of [8] on sparse graphs to provide pseudo-deterministic proofs.

Definition 5.9. We say a graph property is a formula $Q_1 x_1 \in X_1 Q_2 x_2 \in X_2 ... Q_k x_k \in X_k \psi$, where ψ is a quantifier-free formula on edge predicates and the model checking problem for a graph property is to determine whether the property holds for a given graph.

Theorem 5.10 ([8])**.** *If a formula with k does not have the form $\exists^{k-1}\forall$, then the model checking problem for the formula can be solved in co-nondeterministic time m^{k-2} where m is the number of edges in the graph.*

Theorem 5.11 ([8])**.** *The deterministic complexity of model checking a k-quantifier formula is $O(m^{k-1})$.*

Theorem 5.12. *If a formula does not have the form $\exists^{k-1}\forall$, there exists a $(O(m^{k-2}), O(m^{k-1}))$ pseudo-deterministic proof for finding a setting to the first set of existential quantifiers of that formula.*

Proof. If the first i quantifiers are $\exists$, then we can find $x_1, ..., x_i$ such that $Q_{i+1}x_{i+1}...Q_k x_k \psi(x_1, ..., x_i)$ nondeterministically in time $O(m^{k-i})$ for any $1 \leq j \leq i$, and we can check for any $1 \leq j \leq i$ that $\exists x'_j < x_j Q_{j+1}x_{j+1}...Q_k x_k \psi(x_1, ..., x_{j-1})$ in co-nondeterministic time $O(m^{k-2})$ by setting $X'_j = X_j \cap \{x | x < x_j\}$. For both of these checks, the prover has to solve a model checking problem with at most k quantifiers, which has complexity $O(m^{k-1})$. This shows that there is a $O(m^{k-2})$ pseudo-deterministic proof where the prover runs in time $O(m^{k-1})$ for finding a setting to the first set of existential quantifiers of a formula, if the formula does not have the form $\exists^{k-1}\forall$. □

5.4 Problems equivalent to All-Pairs Shortest Path

The All-Pairs Shortest Path problem has been the focus of much research in fine-grained complexity. It has been shown by [39, 38] that many problems related to graphs reduce to the All-Pairs Shortest Path problem and vice versa, so finding a faster algorithm for any one of these problems would yield a fast algorithm for a host of graph problems. [8] shows that the Zero Weight Triangle problem, which is equivalent to the All-Pairs Shortest Path problem under subcubic reductions [38], has a $O(n^{3-\epsilon})$ co-nondeterministic algorithm, which is faster than all known deterministic algorithms. We use this to construct a pseudo-deterministic proof for the Zero Weight Triangle problem.

Definition 5.13. The Zero Weight Triangle problem is given a graph $G = (V, E)$ and edge weights $e(i, j)$, find $i, j, k \in V$ such that $e(i, j) + e(i, k) + e(j, k) = 0$.

Theorem 5.14 ([8])**.** *The Zero Weight Triangle problem has a nondeterministic proof and a co-nondeterministic proof where the verifier runs in time $O(n^{2+\omega/3})$, where ω is the largest number such that matrix multiplication is in time $O(n^{\omega})$.*

Theorem 5.15. *The Zero Weight Triangle problem has an $(\tilde{O}(n^{2+\omega/3}), \tilde{O}(n^3))$ pseudo-deterministic proof.*

Proof. There is an easy reduction from Zero Weight Triangle to Zero Weight Triangle on tripartite graphs. Then, we remove all edges in the first column going from $i' \geq i$ to j, and thus the resulting graph has a triangle with zero weight iff there exists a triangle in the original graph with zero weight and $i' < i$, where i is the smallest vertex in the claimed lexicographically first zero weight triangle. A similar argument as the argument showing the prover for 3-SUM runs in randomized time $\tilde{O}(n^2)$ shows that the prover for the pseudo-deterministic proof of Zero Weight Triangle runs in randomized time $\tilde{O}(n^3)$. By Lemma 3.3, this implies that Zero Weight Triangle has a pseudo-deterministic proof where the verifier runs in time $\tilde{O}(n^{2+\omega/3})$ and the prover runs in $\tilde{O}(n^3)$. □

6 Conclusions and Open Problems

We defined the notion of doubly-efficient pseudo-deterministic proofs and gave a number of examples of search problems for which we showed doubly-efficient pseudo-deterministic proofs. In all of these cases, the verifier runs faster than the best known probabilistic algorithm for the problem which can offer significant improvements for settings in which a more powerful computer (cloud, special purpose device, centralized authority) can perform the computation first and prove it to a significantly less powerful user. In all these cases the prover's computation increases polynomially from what is necessary to solve the problem without need for a canonical solution. An interesting problem would be show that this is true in general. Namely, that for any doubly-efficient pseudo-eterministic proof the computation of the prover need be no more than whats necessary to find the canonical solution. Finally, we remark that in all the cases we treated, the canonical solution was the lexicographically smallest (or largest as in the LP case) but other canonical solutions are possible.

Acknowledgments

We would like to thank Andrea Lincoln for preliminary discussion on pseudo-deterministic proofs for fine-grained complexity. Dhiraj Holden was supported by NSF MACS - CNS-1413920.

References

[1] László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697. ACM, 2016.

[2] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M Reischuk. Adsnark: Nearly practical and privacy-preserving proofs on authenticated data. In *2015 IEEE Symposium on Security and Privacy*, pages 271–286. IEEE, 2015.

[3] Michael Backes, Dario Fiore, and Raphael M Reischuk. Verifiable delegation of computation on outsourced data. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 863–874. ACM, 2013.

[4] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Annual Cryptology Conference*, pages 90–108. Springer, 2013.

[5] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 781–796, 2014.

[6] Michele Borassi, Pierluigi Crescenzi, and Michel Habib. Into the square: On the complexity of some quadratic-time solvable problems. *Electronic Notes in Theoretical Computer Science*, 322:51–67, 2016.

[7] Benjamin Braun, Ariel J Feldman, Zuocheng Ren, Srinath Setty, Andrew J Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 341–357. ACM, 2013.

[8] Marco L Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 261–270. ACM, 2016.

[9] Timothy M Chan. More logarithmic-factor speedups for 3sum,(median,+)-convolution, and some geometric 3sum-hard problems. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 881–897. SIAM, 2018.

[10] Alessandro Chiesa, Eran Tromer, and Madars Virza. Cluster computing in zero knowledge. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 371–403. Springer, 2015.

[11] Michael Cohen, Yin-Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *CoRR*, abs/1810.07896, 2018.

[12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 90–112. ACM, 2012.

[13] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy*, pages 253–270. IEEE, 2015.

[14] Mark De Berg, Marko M de Groot, and Mark H Overmars. Perfect binary space partitions. *Computational geometry*, 7(1-2):81–91, 1997.

[15] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 844–855. ACM, 2014.

[16] Anka Gajentaan and Mark H Overmars. On a class of o(n^2) problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

[17] 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.

[18] Michel Goemans. 18.415/6.854 advanced algorithms lecture 11, 2008. URL:`https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-854j-advanced-algorithms-fall-2008/lecture-notes/lec11.pdf`.

[19] Oded Goldreich and Guy N Rothblum. Simple doubly-efficient interactive proof systems for locally-characterizable sets. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[20] Shafi Goldwasser, Ofer Grossman, and Dhiraj Holden. Pseudo-deterministic proofs. In *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[21] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[22] Ahmed E Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F Sayed, Elaine Shi, and Nikos Triandopoulos. {TRUESET}: Faster verifiable set computations. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 765–780, 2014.

[23] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE, 2013.

[24] Mihai Pătraşcu and Ryan Williams. On the possibility of faster sat algorithms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1065–1075. SIAM, 2010.

[25] Omer Reingold, Guy N Rothblum, and Ron D Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 49–62. ACM, 2016.

[26] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[27] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 71–84. ACM, 2013.

[28] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 253–268, 2012.

[29] Srinath TV Setty, Richard McPherson, Andrew J Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS*, volume 1, page 17, 2012.

[30] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO (2)*, pages 71–89, 2013.

[31] Justin Thaler, Mike Roberts, Michael Mitzenmacher, and Hanspeter Pfister. Verifiable computation with massively parallel interactive proofs. In *HotCloud*, 2012.

[32] Pravin Vaidya. Speeding-up linear programming using fast matrix multiplication. In *FOCS*, 1989.

[33] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[34] Victor Vu, Srinath Setty, Andrew J Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 223–237. IEEE, 2013.

[35] Riad S Wahby, Srinath TV Setty, Zuocheng Ren, Andrew J Blumberg, and Michael Walfish. Efficient ram and control flow in verifiable outsourced computation. In *NDSS*, 2015.

[36] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, 2015.

[37] Ryan Williams. Faster decision of first-order graph properties. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 80. ACM, 2014.

[38] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 645–654. IEEE, 2010.

[39] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal on Computing*, 42(3):831–854, 2013.