A stateful implementation of a random function supporting parity queries over hypercubes

Andrej Bogdanov and Hoeteck Wee

Computer Science Division
University of California, Berkeley
{adib,hoeteck}@cs.berkeley.edu

Abstract. Motivated by an open problem recently suggested by Goldreich et al., we study truthful implementations of a random binary function supporting compound XOR queries over sub-cubes of the hypercube $\{0,1\}^n$. We introduce a relaxed model of an implementation, which we call a *stateful* implementation, and show how to implement the desired specification in this model. The main technical construction is an algorithm for detecting linear dependencies between n dimensional hypercubes, viewed as characteristic vectors in $\mathbb{F}_2^{\{0,1\}^n}$. Using coding theoretic techniques, we first exhibit a randomized algorithm for detecting such dependencies. We then show how a recent approach by Raz and Shpilka for polynomial identity testing in non-commutative models of computation can be applied to obtain a deterministic algorithm.

1 Introduction

In a recent paper, Goldreich, Goldwasser and Nussboim [3] initiated the study of efficient pseudorandom implementations of *huge random objects*—objects so big that they cannot be represented using bounded resources, such as randomness or time (in particular, these objects have size that is exponential in the running time of the applications), but for which we can obtain approximations good enough for many algorithmic and cryptographic applications. A celebrated example from cryptography is the construction of a pseudo-random function from any one way function [2]: Even though a truly random boolean function on n input bits cannot be specified by fewer than 2^n random bits, in many cryptographic applications this infeasible object can be approximated by a *pseudo-random function* that can be specified using only $\mathrm{poly}(n)$ random bits and evaluated on arbitrary inputs in $\mathrm{poly}(n)$ time.

1.1 Stateful and Stateless Implementations

Since the work of Goldreich et al. is somewhat motivated by cryptography, they consider only "stateless" implementations of huge random objects. An implementation refers to a polynomial-time (oracle) machine that computes the huge object; in the case of a random boolean function, the machine takes as input a

string of n bits and outputs the value of the function at that point. In a stateless implementation I, the answer to a query posed to I cannot depend on queries previously seen by I and their answers. This property of the implementation is often important in cryptographic settings, where multiple parties share the same copy of the huge object in question. For example, a common technique in cryptography is to design protocols in the random oracle model, where each party has access to the same infinite sequence of random bits. To obtain an implementation of the protocol, one replaces the random oracle with a pseudo-random function. As we want the parties in the protocol to share the same pseudo-random function, it is important that the implementation of the random oracle by a pseudo-random function be independent of the queries seen by a particular party in the protocol; namely, the implementation must be stateless.

In addition to cryptography, Goldreich et al. also consider algorithmic applications of huge random objects. For example, they imagine a scenario where one wants to run experiments on, say, random codes. They observe that global properties of these codes, such as having large minimum distance, may not be preserved when the randomness of the code is replaced by a pseudo-random generator. This leads to the problem of implementing huge random objects that are guaranteed to preserve a certain property, such as codes with good minimum distance.

Unlike in the cryptographic setting, it is not clear that a stateless implementation (the only type allowed by the model of Goldrech et al.) gives any advantage over a "stateful" one. In other words, it may be possible to do more by generating the desired huge random object "on the fly" rather than subscribing to an implementation predetermined by the random tape of our machine. In particular, it would be interesting to know whether there exists a natural specification that allows a stateful implementation but not a stateless one. We suspect that the specification considered in this paper, suggested for study by Goldreich et al.—a random boolean function supporting XOR queries over hypercubes—may provide a separation between stateful and stateless *perfect implementations in the random oracle model*.

1.2 Random Functions Supporting Complex Queries

Goldreich et al. observe that, assuming the existence of one-way functions, if a specification can be close-implemented[1] in the random oracle model, then it can also be implemented by an ordinary probabilistic polynomial-time machine. Moreover, this transformation preserves truthfulness[2]: Namely, to obtain a truthful pseudo-implementation of a huge random object, it is sufficient to construct

[1] In fact, it is sufficient that the specification be pseudo-implementable. Note that the terms close-implementable, pseudo-implementable and truthful are technical terms defined in [3].

[2] Intuitively, truthfulness requires that an implementation of Type T objects generates only objects of Type T. In particular, a random function is not a truthful implementation of a random permutation even though they are indistinguishable to a computationally bounded adversary.

such an implementation in the random oracle model. This is a common technique in cryptography, used among other things in the construction of pseudo-random permutations [4]. Though the transformation is only shown to hold for stateless implementations, we observe that it also works for stateful ones (this is because the definition of pseudo-randomness in the context of pseudo-random functions allows from stateful adversaries).

In particular, this observation implies that random functions have a trivial truthful pseudo-implementation. However, one may ask whether it is possible to truthfully implement random functions supporting queries beyond evaluation on arbitrary inputs. One variant proposed by Goldreich et al. asks for the implementation of a random function $f : \{0,1\}^n \to \{0,1\}$, augmented with queries regarding the XOR of the values of f on arbitrary *intervals* of $\{0,1\}^n$ (with respect to the lexicographic ordering of n bit strings.) Note that a trivial implementation of f in the random oracle model cannot hope to answer such queries efficiently, as they may involve XORing exponentially many bits. However, Goldreich et al. show how, using a suitable data structure, one can obtain a stateless perfect implementation of f that answers queries in $O(n^2)$ time. As perfect implementations are always truthful, this construction yields a truthful pseudo-implementation by an ordinary machine.

We observe that a simpler construction for implementing random functions supporting interval-XOR queries with $O(n)$ running time can be achieved as follows: let $f' : \{0,1\}^n \to \{0,1\}$ be the random oracle, and return $f'(\alpha-1) \oplus f'(\beta)$ as the answer to the query (α, β), corresponding to the value $\bigoplus_{\alpha \le x \le \beta} f(x)$. Here, $\alpha-1$ denotes the n-bit binary string that immediately precedes α in lexicographic order, and we specify $f'(0^n - 1) = 0$. The underlying idea is a simple change of basis: instead of specifying a random function f by its values at all $x \in \{0,1\}^n$, we specify the value of f using the values $\bigoplus_{y \le x} f(y)$. Note that this implementation makes only 2 queries into the random oracle per interval-XOR query, which is optimal (in an amortized sense). Unfortunately, this construction, unlike that by Goldreich et al., does not yield a truthful close-implementation of random functions supporting any symmetric interval query.

As a follow-up to their work on interval queries, Goldreich et al. propose the following more general question:

Open Problem. [3] Provide a truthful close-implementation in the random oracle model of the following specification. The specification machine defines a random function $f : \{0,1\}^n \to \{0,1\}$, and answers queries that succinctly describe a set S, taken from a specific class of sets, with the value $\bigoplus_{x \in S} f(x)$. A natural case is the class of sub-cubes of $\{0,1\}^n$; that is, a set S is specified by a pattern σ in $\{0,1,*\}^n$ such that S is the set of points in $\{0,1\}^n$ that match the pattern σ.

We suspect that the technique of Goldreich et al. for implementing a random function supporting interval queries does not extend to the case of hypercube queries, though we have not been able to show a negative result confirming our intuition. Instead, we show how to obtain a *stateful* perfect-implementation of this specification in the random oracle model.

1.3 Main Contributions

Our main contributions are the following:

1. We propose a notion of stateful implementations of huge random objects and reduce the problem of constructing a stateful implementation of a random binary function supporting compound XOR queries over sub-cubes of the hypercube $\{0,1\}^n$ to an algorithmic problem of hypercube linearity testing.
2. We then present two algorithms for hypercube linearity testing: a randomized algorithm using coding theoretic techniques, and a deterministic based on non-commutative polynomial identity testing. It follows from the first algorithm that there is a stateful close implementation of the afore-mentioned specification in the random oracle model, and from the second, a stateful perfect implementation.

In fact, the second algorithm subsumes the first, but we still include the latter as we feel that the technique used is fairly insightful and the analysis can in fact be used to construct quantum states of provably superpolynomial tree size.

2 Preliminaries

Let $a \in \{0,1,*\}^n$. The *hypercube*[3] $H(a)$ is the set of all $x \in \{0,1\}^n$ that match the string a, namely such that

$$x[i] = \begin{cases} 0, & \text{if } a[i] = 0 \\ 1, & \text{if } a[i] = 1 \\ 0 \text{ or } 1, & \text{if } a[i] = *. \end{cases}$$

As in Goldreich et al., we specify the huge random object in question by a computationally unbounded probabilistic Turing machine that halts with probability one. The huge random object is determined by the input-output relation of this machine when the random tape is selected uniformly at random from $\{0,1\}^\infty$.

A random function supporting XOR queries on hypercubes
INPUT A query $a \in \{0,1,*\}^n$
RANDOM TAPE A sequence of functions $f_1, f_2, \ldots$, where $f_k : \{0,1\}^k \to \{0,1\}$
OUTPUT The value $\bigoplus_{x \in H(a)} f_{|a|}(x)$

We are interested in efficient implementations of this specification, namely ones that can be obtained by Turing machines that run in time polynomial in the length of the input.

[3] In [3] and in the introduction, we use the term sub-cubes of the hypercube $\{0,1\}^n$.

2.1 Stateful Implementations

Following Goldreich et al., we say machine I is a *(stateful) implementation* specification S with respect to a machine M of that makes queries to a protocol party, if (1) On the qth query x, I runs in time polynomial in q and x, and (2) The distribution D_S of the transcript $M(1^n) \leftrightarrow S$ is indistinguishable from the distribution D_I of $M(1^n) \leftrightarrow I$. Specifically:

1. If D_S and D_I are identical for every M, we say that I *perfectly implements* S;
2. If D_S and D_I have negligible ($n^{\omega(1)}$) statistical difference for all M that make poly(n) queries, we say that I *closely implements* S;
3. If D_S and D_I are indistinguishable for all M that run in polynomial time, we say that I *pseudo-implements* S.

An implementation I is *truthful* with respect to S if for every sequence of queries $x_1, \ldots, x_q$, the support of the distribution $I(x_1), \ldots, I(x_q)$ is contained in the support of $S(x_1), \ldots, S(x_q)$; namely, if the implementation never provides answers that are inconsistent with the specification. Note that perfect implementations are always truthful. An oracle Turing machine $I^?$ is an implementation of S *in the random oracle model* if the distribution of $M(1^n) \leftrightarrow S$ is indistinguishable from the distribution of $M(1^n) \leftrightarrow I^R$ over a random oracle R. As for stateless implementations (see Theorem 2.9 of [3]), we have the following:

Proposition 1. *Suppose that one-way functions exist. Then any specification that has a pseudo-implementation in the random oracle model also has a pseudo-implementation by an ordinary machine. Moreover, if the former implementation is truthful then so is the latter.*

Since perfect implementations are always truthful, for our purposes it will be sufficient to provide a perfect implementation of a random function supporting queries on hypercubes in the random oracle model. The heart of this implementation consists of an algorithm for the problem of hypercube linearity testing, or Hypercube-Lin, which we describe next.

2.2 Reduction to Hypercube Linearity Testing

We call collection of hypercubes $H_1, \ldots, H_m \subseteq \{0,1\}^n$ *linearly dependent* if there exists a non-empty set $S \subseteq \{1, 2, \ldots, m\}$ such that for all points $x \in \{0,1\}^n$, x is contained in an even number of hypercubes amongst the subset of hypercubes $\{H_i \mid i \in S\}$. For any such set S, we write $\sum_{i \in S} H_i = 0$. Equivalently, we may view each hypercube H_i as a vector in $\mathbb{F}_2^{\{0,1\}^n}$, where $\mathbb{F}_2 = \{0,1\}$ is the two element field and $H_i[x] = 1$ if $x \in H_i$, and 0 otherwise. In this notation, linear independence between hypercubes translates into linear independence of the corresponding vectors.

Hypercube-Lin: Given q strings $a_1, \ldots, a_q \in \{0,1,*\}^n$, accept iff the hypercubes $H(a_1), \ldots, H(a_q)$ are linearly dependent.

Note that standard techniques for testing linear independence, such as Gaussian elimination, do not apply directly for this problem because we are dealing with vectors whose length is exponential in the size of the input. However, we will still be able to show the following:

Theorem 1. *There is a deterministic polynomial-time algorithm for the problem* Hypercube-Lin.

In fact, we will begin with a randomized algorithm for Hypercube-Lin, which nicely illustrates the coding theoretic nature of the hypercube linearity testing problem. We then argue that the test performed by the algorithm can be viewed, in some sense, as an application of polynomial identity testing. Even though we don't know, in general, how to derandomize polynomial identity testing, in our case the derandomization can be performed using a recent identity testing algorithm for non-commutative formulas of Raz and Shpilka [5].

Theorem 2. *There exists a stateful perfect implementation of a random function supporting XOR queries on hypercubes in the random oracle model.*

Proof. Let A be the algorithm from Theorem 1, and R be the random oracle. First, we consider the following (promise) search problem:

Input Strings $a_1, \ldots, a_q \in \{0,1,*\}^n$ and $b \in \{0,1,*\}^n$, such that the hypercubes $H(a_1), \ldots, H(a_q)$ are linearly independent
Problem If $H(b)$ is linearly independent from $H(a_1), \ldots, H(a_q)$, output $\varnothing$. Otherwise, output coefficients $c_1, \ldots, c_q \in \mathbb{F}_2$ such that $H(b) = \sum_{i=1}^{q} c_i H(a_i)$.

It is not difficult to see that, by a self reduction argument, we can obtain a polynomial time algorithm A' for this problem using black box access to A (in fact, $2q$ invocations of A suffices). With this in hand, we implement a random function supporting XOR queries on hypercubes as follows: After seeing queries $a_1, \ldots, a_{q-1}$, the implementation keeps track of a subset of queries $\{a_k : k \in B\}$, where $B \subseteq [q-1]$ is chosen such that the subset of hypercubes $\{H(a_k) : k \in B\}$ form a basis for the set of vectors $\{H(a_k) \in \mathbb{F}_2^{\{0,1\}^n} : k \in [q-1]\}$. On query a_q, we run the algorithm A' on inputs $\{a_k : k \in B\}$ and a_q. If the algorithm returns $\varnothing$, then we return $R(|B|+1)$, which is a fresh random bit, and add a_q to the set B. Otherwise, the algorithm A' outputs coefficients $c_1, \ldots, c_{q-1} \in \mathbb{F}_2$, and we return the value $\sum_{i=1}^{|B|} c_i R(i)$.

We show this is a perfect implementation, by induction on q. Let us assume that the specification transcript and implementation transcript are statistically indistinguishable after $q-1$ queries. At query q, there are two possibilities: If $H(a_q)$ is linearly dependent in $H(a_1), \ldots, H(a_{q-1})$, then both in the implementation and in the specification the answer to the qth query is determined by the previous answers, so by the inductive hypothesis the new transcripts are indistinguishable. If $H(a_q)$ is linearly independent in $H(a_1), \ldots, H(a_{q-1})$, then the answer to the qth query in the specification is statistically independent from all previous answers. By construction, this is also true in the implementation, so again, the new transcripts are indistinguishable.

3 Algorithms for Hypercube Linearity Testing

In this section, we present two algorithms for HYPERCUBE-LIN, thereby completing our implementation of a random function supporting XOR queries on hypercubes.

3.1 A Randomized Algorithm for HYPERCUBE-LIN

Let $\mathbb{F} = \mathbb{F}_{2^s}$ be a field of characteristic 2 of size 2^s, where $s = \lceil \log 3n \rceil$. For each $a \in \{0, 1, *\}^n$, we define a polynomial $p_a = z_1 z_2 \cdots z_n$ over $\mathbb{F}[x_1, y_1, \ldots, x_n, y_n]$, where:

$$z_i = \begin{cases} x_i & \text{if } a_i = 0 \\ y_i & \text{if } a_i = 1 \\ x_i + y_i & \text{if } a_i = * \end{cases}$$

Note that $H(a)$ is a homogeneous polynomial of total degree exactly n, and that $p_a = \sum_{x \in H(a)} p_x$.

Lemma 1. *Let $\mathbb{F}$ be any field of characteristic two. Then $H(a_1) + \cdots + H(a_m) = 0$ if and only if $p_{a_1} + \cdots + p_{a_m}$ is the zero polynomial in $\mathbb{F}[x_1, y_1, \ldots, x_n, y_n]$.*

Proof. Each point x in $\{0,1\}^n$ is represented by a unique monomial p_x from $\mathbb{F}[x_1, y_1, \ldots, x_n, y_n]$. Therefore, each point x in $\{0,1\}^n$ appears an even number of times in $H(a_1), \ldots, H(a_m)$ iff each of the corresponding monomials p_x has an even integer coefficient in $p_{a_1} + \cdots + p_{a_m}$. In addition, since $\mathbb{F}$ has characteristic 2, the latter condition is equivalent to $p_{a_1} + \cdots + p_{a_m}$ being identically zero.

We may now define a binary encoding $C_{H(a)}$ of hypercubes $H(a)$, which is obtained by concatenating the Reed-Muller code associated with the polynomial p_a with the Hadamard code. More precisely, given any $a \in \{0, 1, *\}$, we define $C_{H(a)} : \{0,1\}^{(2n+1)s} \to \{0,1\}$ as follows:

$$C_{H(a)}(x_1, y_1, \ldots, x_n, y_n, \tau) = \langle\, p_a(x_1, y_1, \ldots, x_m, y_m), \tau \,\rangle$$

using the first $2ns$ bits of the input to $C_{H(a)}$ to pick $x_1, y_1, \ldots, x_n, y_n \in \mathbb{F}$, and the remaining s bits to pick $\tau \in \{0,1\}^s$.

Lemma 2. *$\{C_{H(a)} \mid a \in \{0,1,*\}^n\}$ is a binary encoding of hypercubes in $\{0,1\}^n$ with the following properties:*

1. *(large distance) It has relative distance $1/3$.*
2. *(locally encodable) There is a $O(ns \log s)$ algorithm that computes $C_{H(a)}(r)$ on input a and $r \in \{0,1\}^{(2n+1)s}$.*
3. *(linearity) For all $a, a' \in \{0,1,*\}^n$, $C_{H(a)+H(a')} = C_{H(a)} + C_{H(a')}$.*

Proof. By the Schwartz-Zippel Lemma [6, 7], for any $a \neq a' \in \{0,1,*\}^n$, p_a and $p_{a'}$ evaluate to different values on at least a 2/3 fraction of values in $\mathbb{F}^{2n}$. Upon concatenating with the Hadamard code, the minimum relative distance becomes 1/3. Next, observe that we can write down p_a and evaluate p_a at any input in time $O(ns \log s)$. It follows that we also compute $C_{H(a)}(r)$ in time $O(ns \log s)$. Finally, linearity follows from the fact that both the encoding as polynomials and the Hadamard code are linear.

Randomized algorithm for HYPERCUBE-LIN

1. Fix $\ell = O(q)$. Choose $r_1, \ldots, r_\ell \in \{0,1\}^{(2n+1)s}$ uniformly at random.
2. Construct the $q \times \ell$ matrix M over $\mathbb{F}_2$, where $M_{ij} = C_{H(a_i)}(r_j)$, that is:

$$M = \begin{pmatrix} C_{H(a_1)}(r_1) & C_{H(a_1)}(r_2) & \ldots & C_{H(a_1)}(r_\ell) \\ C_{H(a_2)}(r_1) & C_{H(a_2)}(r_2) & \ldots & C_{H(a_2)}(r_\ell) \\ \vdots & \vdots & \ddots & \vdots \\ C_{H(a_q)}(r_1) & C_{H(a_q)}(r_2) & \ldots & C_{H(a_q)}(r_\ell) \end{pmatrix}$$

3. Compute the rank of M over $\mathbb{F}_2$. Accept if the rank is less than q; reject otherwise.

Proposition 2. *There is a* coRP*-algorithm for* HYPERCUBE-LIN *running in time* $O(q^3 + q^2 ns \log s)$.

Proof. Fix any $S \subseteq [q]$. It follows from linearity and Lemma 1 that

$$\sum_{i \in S} H(a_i) = 0 \quad \text{if and only if} \quad \sum_{i \in S} C_{H(a_i)}(r) = 0 \quad \forall\, r \in \{0,1\}^{(2n+1)s} \tag{1}$$

Therefore, if $H(a_1), \ldots, H(a_q)$ are linearly dependent, there is some S for which (1) holds. Then, the rows of M identified by S add up to 0, and thus M has rank less than q. On the other hand, if $H(a_1), \ldots, H(a_q)$ are not linearly dependent, then for all $S \subseteq [q]$, $\sum_{i \in S} C_{H(a_i)}$ is not the zero codeword[4], and thus

$$\Pr_{r_1,\ldots,r_\ell}\Big[\textstyle\sum_{i \in S} C_{H(a_i)}(r_j) = 0 \quad \forall\, j = 1, 2 \ldots, \ell\Big] \leq \left(\frac{2}{3}\right)^\ell$$

Now, the probability that M has rank less than q is equal to the probability that there is some $S \subseteq [q]$ for which (1) holds, which by a union bound is at most $2^q \cdot (2/3)^\ell < 1/2$. Finally, computing M takes time $O(q\ell ns \log s)$, and performing Gaussian elimination on M takes time $O(q^2 \ell)$, which yields a total running time of $O(q^3 + q^2 ns \log s)$.

[4] In fact, $\sum_{i \in S} C_{H(a_i)}$ is not necessarily a codeword in the code defined in Lemma 2, but it is still a codeword in the code obtained by concatenating a Reed Muller code with a Hadamard code. Hence, the analysis for the relative distance of the code also shows that $\sum_{i \in S} C_{H(a_i)}$ has relative distance at least 1/3 from the zero codeword.

3.2 A Deterministic Algorithm for Hypercube-Lin

The randomized algorithm for hypercube linearity testing is based on Lemma 1, which allows us to reduce testing a particular dependency between hypercubes to a polynomial identity test. In the actual algorithm, the power of randomness is used twice. First, randomness allows us to efficiently perform the polynomial identity test from Lemma 1. Second, using standard amplification of success probabilities we can take a union bound over the exponentially many possible linear dependencies between hypercubes in the correctness proof for the randomized algorithm.

Even though polynomial identity testing remains notoriously hard to derandomize, Raz and Shpilka [5] recently found a deterministic algorithm that works for certain alternate and restricted models of computation. In particular, their algorithm works for formulas over arbitrary fields in non-commuting variables (i.e., formulas in the noncommutative ring $\mathbb{F}\{x_1, \ldots, x_n\}$), and for $\Sigma\Pi\Sigma$ *circuits*—depth three multilinear arithmetic circuits with a plus gate at the root.

The formal polynomial $p_1 + \ldots + p_m$ in Lemma 1 can be trivially computed by an $\Sigma\Pi\Sigma$ circuit. This immediately gives a deterministic way of testing whether a particular linear relation $H_1 + \ldots + H_m = 0$ is satisfied over $\mathbb{F}_2$. However, we are interested in testing whether any one of the exponentially many relations $c_1 H_1 + \ldots + c_m H_m = 0$ holds, or equivalently, if there exists coefficients $c_1, \ldots, c_m \in \mathbb{F}_2$, not all zero, such that $c_1 p_1 + \ldots + c_m p_m = 0$ as a polynomial in $\mathbb{F}_2[x_i, y_i]$. We will show that a construction, along the lines of Raz and Shpilka's algorithm, works for this problem as well. Instead of trying to come up with a general theorem for testing linear dependencies between polynomials, we will focus on polynomials representing hypercubes, though it may be possible to extend the analysis to a somewhat more general scenario.

Let $\mathbb{F}$ now be an arbitrary finite field, and $p_1, \ldots, p_m$ be polynomials in $\mathbb{F}[x_{ij}]$, where $i \in [n], j \in [m]$ and each $p_k, k \in [m]$ has the following form:

$$p_k(x_{ij}) = \prod_{i=1}^{n} \sum_{j=1}^{m} \alpha_{ij}^k x_{ij}, \tag{2}$$

where $\alpha_{ij}^k \in \mathbb{F}$ are constants. It is easy to check that the polynomials in Lemma 1 are written in this form. We are interested in determining, in time polynomial in m and n, whether there exist coefficients $c_1, \ldots, c_m \in \mathbb{F}$ such that $c_1 p_1 + \ldots + c_m p_m$ is the zero polynomial over $\mathbb{F}$. Since this is a multilinear polynomial, this is equivalent to asking whether $p_1, \ldots, p_m$ are linearly dependent as elements of the vector space V over $\mathbb{F}$ generated by the monomials $x_{1j_1} \ldots x_{nj_n}$, where $j_1, \ldots, j_n$ range over $[m]$.

As in [5], the analysis works by induction on n. When $n = 1$, we have $p_k(x_{1j}) = \sum_{j=1}^{m} \alpha_{1j}^k x_{1j}$, so the vectors $p_k \in V$ are independent if and only if the matrix $M \in \mathbb{F}^{m \times m}$ with $M[j,k] = \alpha_{1j}^k$ has full rank. Using Gaussian elimination this can be checked in, say, $O(m^3 \log |F|)$ time.

We now show how to reduce a problem of degree n to one of degree $n-1$. The trick is to look at the following partial expansion of the polynomials p_k:

$$p_k(x_{ij}) = \sum_{j_1,j_2=1}^{m} \alpha_{1j_1}^k \alpha_{2j_2}^k x_{1j_1} x_{2j_2} \cdot q_k(x_{(3\ldots n)j}),$$

where q_k is given by

$$q_k(x_{(3\ldots n)j}) = q_k(x_{3j}, x_{4j}, \ldots, x_{dj}) = \prod_{i=3}^{n} \sum_{j=1}^{m} \alpha_{ij}^k x_{ij}.$$

Now consider a linear combination $P(x_{ij}) = \sum_{k=1}^{m} c_k p_k(x_{ij})$. We can expand this as

$$P(x_{ij}) = \sum_{j_1,j_2=1}^{m} x_{1j_1} x_{2j_2} \sum_{k=1}^{m} \alpha_{1j_1}^k \alpha_{2j_2}^k \cdot c_k q_k(x_{(3\ldots n)j}).$$

From this expansion, we see that $P \equiv 0$ in $\mathbb{F}[x_{ij}]$ if and only if for all $j_1, j_2 \in [m]$:

$$\sum_{k=1}^{m} \alpha_{1j_1}^k \alpha_{2j_2}^k \cdot c_k q_k(x_{(3\ldots n)j}) \equiv 0 \text{ in } \mathbb{F}[x_{(3\ldots n)j}]. \tag{3}$$

Consider the matrix $M \in \mathbb{F}^{m^2 \times m}$, whose rows are indexed by pairs $(j_1, j_2) \in [m] \times [m]$ such that $M[(j_1, j_2), k] = \alpha_{1j_1}^k \alpha_{2j_2}^k$. Choose a subset S of m rows such that the rows indexed by S span the row space of M. Then the constraints (3) are all satisfied if and only if

$$\text{For all } s \in S,\ \sum_{k=1}^{m} M[s,k] \cdot c_k q_k(x_{(3\ldots n)j}) \equiv 0 \text{ in } \mathbb{F}[x_{(3\ldots n)j}]. \tag{4}$$

We now want to rewrite the set of constraints (4) as a single constraint. For this, we introduce m additional formal variables y_s with $s \in S$. Constraints (4) are satisfied if and only if

$$\sum_{s \in S} y_s \sum_{k=1}^{m} M[s,k] \cdot c_k q_k(x_{(3\ldots n)j}) \equiv 0 \text{ in } \mathbb{F}[y_s, x_{(3\ldots n)j}]. \tag{5}$$

Finally, let

$$r_k(y_s, x_{(3\ldots n)j}) = \Big(\sum\nolimits_{s \in S} M[s,k] y_s\Big) q_k(x_{(3\ldots n)j}),$$

and note that constraint (5) is satisfied if and only if $\sum_{k=1}^{m} c_k r_k(y_s, x_{(3\ldots n)j}) \equiv 0$ in $\mathbb{F}[y_s, x_{(3\ldots n)j}]$. To summarize:

Lemma 3. *The polynomials $p_1, \ldots, p_m$ are linearly independent in $\mathbb{F}[x_{ij}]$ if and only if the polynomials $r_1, \ldots, r_m$ are linearly independent in $\mathbb{F}[y_s, x_{(3\ldots n)j}]$.*[5]

[5] Moreover, the p_i satisfy a particular dependency $\sum c_i p_i \equiv 0$ if and only if the r_i satisfy the same dependency. This can be used to find linear dependencies between hypercubes, which makes the decision to search reduction in the proof of Proposition 1 unnecessary.

Now we can apply the induction hypothesis to the polynomials r_k, which have the prescribed form (2). The bottleneck in the running time of the algorithm is computing the linearly independent set S, which can be done in $O(m^4 \log |\mathbb{F}|)$ time by Gaussian elimination. This yields a total running time of $O(m^4 n \log |\mathbb{F}|)$, and concludes the proof of Theorem 1.

4 Extensions

4.1 A Connection with Quantum Computation

The following problem was brought to our attention by Aaronson in his recent work addressing skepticism of quantum computing [1]. The construction below implies the existence of explicit (constructible in deterministic polynomial-time) quantum states of provably superpolynomial tree size. The argument is similar to the proof of Proposition 2.

Proposition 3. *For every $\delta < 1/4$, there is a polynomial-time constructible $k \times n$ binary matrix M with $k = n^{\delta}$, such that a random $k \times k$ submatrix of M has rank at least $k - 1$ with constant probability.*

Aaronson's original argument requires M to have full rank with constant probability, but it is not difficult to see that his argument applies even if M has only rank $k - 1$ with constant probability.

Proof. Fix $\epsilon > 0$ and choose k_0 such that $k = k_0 \log k_0^{2+\epsilon}$. Consider the concatenation of a Reed-Solomon code with codeword size $k_0^{2+\epsilon}$ (over smallest possible alphabet size) with a Hadamard code. The resulting binary code C has length at most $n = k^{4+2\epsilon}$ and relative distance $1/2 - \eta$, where $\eta = 1/k^{1+\epsilon-o(1)}$. Pick a basis $e_1, \ldots, e_k$ of $\{0,1\}^k$ and take M to be the $k \times n$ matrix whose ith row is the codeword $C(e_i)$.

Now consider choosing a random $k \times k$ submatrix of M. Since $k = O(n^{1/4})$, with probability $1 - o(1)$, this is equivalent to choosing an independent set of k columns $M_1, \ldots, M_k$ of M with repetition. For any nonzero vector $x \in \{0,1\}^k$, the probability that all dot products $x^T M_1, \ldots, x^T M_k$ vanish is at most $(1/2 + \eta)^k$. It follows that the expected number of nonzero x for which all these dot products vanish is at most $2^k \cdot (1/2 + \eta)^k < \exp(2k^{-\epsilon}) = 1 + o(1)$. By Markov's inequality, the number of such x is less than two with probability at least, say, $1/3$. If this is the case, then the columns of M satisfy at most one linear relation, so that the rank of M is at most $k - 1$.

4.2 Towards a Stateless Implementation?

We discuss our conjecture regarding a possible separation between stateful and stateless perfect implementations in the random oracle model. We say that a set S of vectors in $\mathbb{F}_2^{2^n}$ admits an *efficient basis* B where B is a (standard) basis for $\mathbb{F}_2^{2^n}$ if every vector $v \in S$ can be written as the sum of $\text{poly}(n)$ vectors in B and the basis representation can be computed in $\text{poly}(n)$ time.

Proposition 4. *If S admits an efficient basis, then there is a perfect (stateless) implementation of a random function $f : \{0,1\}^n \to \{0,1\}$, and answers queries that succinctly describe a set S, taken from a specific class of sets, with the value $\bigoplus_{x \in S} f(x)$.*

It is easy to see that intervals of $\{0,1\}^n$ admit an efficient basis, namely the collection of vectors $\{v_i \in \mathbb{F}_2^{2^n}, i = 1, 2, \ldots, 2^n\}$ where v_i is the vector whose first i positions are ones and whose remaining positions are zeros. This observation underlies our construction for implementing random functions support interval-XOR queries in Section 1.2. On the other hand, we do not know if the hypercubes of $\{0,1\}^n$ admit an efficient basis.

Acknowledgements

We thank Luca Trevisan and the anonymous referees for comments on an earlier version of this paper.

References

[1] Scott Aaronson. Multilinear formulas and skepticism of quantum computing. In *Proceedings of the 36th ACM Symposium on Theory of Computing*, 2004.

[2] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):210–217, 1986.

[3] Oded Goldreich, Shafi Goldwasser, and Asaf Nussboim. On the implementation of huge random objects. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science*, pages 68–79, 2003. Preliminary full version at `http://www.wisdom.weizmann.ac.il/~oded/p_toro.html`.

[4] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudoranom functions. *SICOMP*, 17:373–386, 1988.

[5] Ran Raz and Amir Shpilka. Deterministic polynomial identity testing in non commutative models. In *Proceedings of the 17th Conference on Computational Complexity*, 2004. To appear.

[6] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

[7] R. E. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of EUROSAM 79*, pages 216–226, 1979.