On the Implementation of Huge Random Objects*

Oded Goldreich† Shafi Goldwasser‡ Asaf Nussboim§

December 29, 2009

Abstract

We initiate a general study of the feasibility of implementing (huge) random objects, and demonstrate its applicability to a number of areas in which random objects occur naturally. We highlight two types of measures of the quality of the implementation (with respect to the desired specification): The first type corresponds to various standard notions of indistinguishability (applied to function ensembles), whereas the second type is a novel notion that we call truthfulness. Intuitively, a truthful implementation of a random object of Type T must (always) be an object of Type T, and not merely be indistinguishable from a random object of Type T.

Our formalism allows for the consideration of random objects that satisfy some fixed property (or have some fixed structure) as well as the consideration of objects supporting complex queries. For example, we consider the truthful implementation of random Hamiltonian graphs as well as supporting complex queries regarding such graphs (e.g., providing the next vertex along a fixed Hamiltonian path in such a graph).

Keywords: Pseudorandomness, Random Graphs, Random Codes, Random Functions, monotone graph properties, random walks on regular graphs.

*An extended abstarct of this work appeared in the proceedings of the *44th FOCS*, pages 68–79, 2003.

†Department of Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel. Partially supported by the MINERVA Foundation (Germany) and by the Israel Science Foundation (grant No. 1041/08). Email: oded@wisdom.weizmann.ac.il

‡Department of Mathematics and Computer Science of the Weizmann Institute of Science and the Laboratory for Computer Science of MIT. Email: shafi@theory.csail.mit.edu

§Shmuel Hanagid 5, Tel-Aviv, Israel. Email: assaf.nussbaum@gmail.com

Contents

1 Introduction

Suppose that you want to run some experiments on random codes (i.e., subsets of $\{0,1\}^n$ that contain $K = 2^{\Omega(n)}$ strings). You actually take it for granted that the random code will have *large* (i.e., linear) *distance*, because you are willing to discard the negligible probability that a random code will not have a large distance. Suppose that you want to be able to keep succinct representations of these huge codes and/or that you want to generate them using few random bits. A natural idea that comes to mind is using pseudorandom functions [20] in order to efficiently generate and store representations of these codes; that is, using the pseudorandom function $f : [K] \to \{0,1\}^n$, one can define the code $C_f = \{f(i) : i \in [K]\}$, and efficiently produce codewords of C_f. *But do the codes generated this way have a large distance?*

The point is that having a large distance is a global property of the code, which in turn is a huge (i.e., $\exp(n)$-sized) object. This global property cannot be decided by looking at polynomially many (i.e., $\text{poly}(n)$-many) codewords, and so its violation cannot be translated to a contradiction of the pseudorandomness of the function. Indeed, the substitution of a random function (or a random code) by a pseudorandom one is not guaranteed to preserve the global property. Specifically, all pseudorandom codes generated as suggested above *may* have small distance.[1]

So, *can we efficiently generate random-looking codes of large distance?* Specifically, can we provide a probabilistic polynomial-time procedure that allows to sample codewords from a code of large distance such that the sampled codewords look as if they were taken from a random code (which, in particular, means that we do not generate linear codes). The answer is essentially positive: see Section 4. However, this is merely an example of the type of questions that we deal with. Another illustrative example is provided by the question of whether it is feasible to generate a random-looking *connected* graph of huge size? Again, the huge graph should look random and be connected, and we cannot obtain this by merely using a random function (see Example 3.5).

The foregoing discussion alludes to the notion of a "truthful" implementation (of a given specification), which will be central to this work. For example, if the specification calls for (random) codes of large distance then the implementation should provide such codes and not arbitrary random-looking codes. However, *even when discarding the question of truthfulness, a fundamental question arises: which types of random objects can be efficiently implemented in the sense that one cannot distinguish the implementation from the corresponding specification.*

We initiate a general study of the feasibility of implementing (huge) random objects. The pivots of this study are the notions of a specification and an implementation (see Section 1.1), where an implementation is related to the specification by appropriate measures of indistinguishability and truthfulness (see Section 1.2). After establishing the basic formalism (in Section 2), we explore several areas in which the study of random objects occurs naturally. These areas include graph theory, coding theory, and cryptography. The bulk of this work provides implementations of various natural random objects, which were considered before in these areas (e.g., the study of random graphs [8]).

1.1 Objects, specifications, and implementations

Our focus is on huge objects; that is, objects that are of size that is exponential in the running time of the applications. Thus, these (possibly randomized) applications may inspect only small portions of the object (in each randomized execution). The object may be viewed as a function (or an oracle), and inspecting a small portion of it is viewed as receiving answers to a small number of adequate queries. For example, when we talk of huge dense graphs, we consider adjacency queries that are vertex-pairs with answers indicating whether or not the queried pair is connected by an edge. When we talk of huge bounded-degree graphs, we consider incidence queries that correspond to vertices with answers listing all the neighbors of the queried vertex.

[1]Indeed, for each function f_s taken from some pseudorandom ensemble $\{f_s : [2^{|s|/10}] \to \{0,1\}^{|s|}\}_s$, it *may* hold that the Hamming distance between $f_s(i_s)$ and $f_s(i_s+1)$ is one, for some i_s that depends arbitrarily on f_s. For example, given a pseudorandom ensemble $\{f_s\}$, consider the ensemble $\{f_{s,i}\}$ such that $f_{s,i}(i) = 0^n$, $f_{s,i}(i+1) = 0^{n-1}1$ and $f_{s,i}(x) = f_s(x)$ for all other x's.

We are interested in classes of objects (or object types), which can be viewed as classes of functions. (Indeed, we are not interested in the trivial case of generic objects, which is captured by the class of all functions.) For example, when we talk of simple undirected graphs in the adjacency predicate representation, we only allow symmetric and non-reflexive Boolean functions. Similarly, when we talk of such bounded-degree graphs in the incident-lists representation, we restrict the class of functions in a less trivial manner (i.e., u should appear in the neighbor-list of v iff v appears in the neighbor-list of u). More interestingly, we may talk of the class of connected (or Hamiltonian) graphs, in which case the class of functions is even more complex. This formalism allows to talk about objects of certain types (or of objects satisfying certain properties). In addition, it captures "complex objects" that support "compound queries" to more basic objects. Indeed, these complex objects are defined in terms of the compound queries that they support (akin to abstract data types that are defined in terms of the supported queries) For example, we may consider an object that answers queries regarding a global property of a Boolean function (e.g., the parity of all the function's values). The queries may also refer to a large number of values of the function (e.g., the parity of all values assigned to arguments in an interval that is specified by the query).

We study probability distributions over classes of objects. Such a distribution is called a specification. Formally, a specification is presented by a *computationally-unbounded* probabilistic Turing machine, where each setting of the machine's random-tape yields a huge object. The latter object is defined as the corresponding input-output relation, and so queries to the object are associated with inputs to the machine. We consider the distribution on functions obtained by selecting the specification's random-tape uniformly. For example, a random N-vertex Hamiltonian graph is specified by a computationally-unbounded probabilistic machine that uses its random-tape to determine such a (random Hamiltonian) graph, and answers adjacency queries accordingly. Another specification may require to answer, in addition to adjacency queries regarding a uniformly selected N-vertex graph, also more complex queries such as providing a clique of size $\log_2 N$ that contains the queried vertex. We stress that the specification is not required to be even remotely efficient (but for sake of simplicity we assume that it is recursive).

Our ultimate goal will be to provide a *probabilistic polynomial-time* machine that implements the desired specification. That is, we consider the *probability distribution* on functions induced by fixing the random-tape of the latter machine in all possible ways. Again, each possible fixing of the random-tape yields a function corresponding to the input-output relation (of the machine per this contents of its random-tape). Thus, an implementation is a probabilistic machine, just as the specification, and it defines a distribution on functions in the same manner. The key difference is that the implementation is a probabilistic polynomial-time machine, whereas the specification is rather arbitrary (or merely recursive).

1.2 Indistinguishability and Truthfulness

Needless to say, the key question is how does the implementation relate to the desired specification; that is, how "good" is the implementation. We consider two aspects of this question. The first (and more standard) aspect is whether one can distinguish the implementation from the specification when given oracle access to one of them. Variants include perfect indistinguishability, statistical-indistinguishability, and computational-indistinguishability.

We highlight a second aspect regarding the quality of implementation: the truthfulness of the implementation with respect to the specification, where being truthful means that any possible function that appears with non-zero probability in the implementation must also appear with non-zero probability in the specification. For example, if the specification is of a random Hamiltonian graph then a truthful implementation must always yield a Hamiltonian graph. Likewise, if the specification is of a random non-Hamiltonian graph then a truthful implementation must always yield a non-Hamiltonian graph. Indeed, these two examples are fundamentally different, because with overwhelmingly high probability a random graph is Hamiltonian. (Thus, a relaxed notion of truthfulness is easy to obtain in the first case but not in the second case.)[2]

Indeed, our presentation highlights the notion of truthfulness, and we justify below the importance that we attach to this notion. Nevertheless, we stress that our work also initiates the study of general

[2]Here we refer to a relaxation of the notion of truthfulness that (only) requires that all but a negligible part of the probability mass of the implementation is assigned to functions that appear with non-zero probability in the specification. An implementation satisfying this relaxation will be called called almost-truthful.

implementations, regardless of truthfulness. That is, we ask which specifications have implementations (which are indistinguishable from them). We also stress that some of our constructions are interesting regardless of their truthfulness.

The meaning of truthfulness. Seeking a truthful implementation of random objects of a given Type T, means aiming at the generation of pseudorandom objects of Type T. That is, we want the generated object to always be of Type T, but we are willing to settle for Type T objects that look as if they are truly random Type T objects (although they are not). This means that we seek *Type T objects* that look like random Type T objects, rather than *objects that look like random Type T objects* although they are not of Type T at all. For example, a random function is not a truthful implementation of a random permutation, although the two look alike to anybody restricted to resources that are polynomially related to the length of the inputs to the function. Beyond the intuitive conceptual appeal of truthfulness, there are important practical considerations.

In general, when one deals (or experiments) with an object that is supposed to be of Type T, one may assume that this object has all the properties enjoyed by all Type T objects. If this assumption does not hold (even if one cannot detect this fact during initial experimentation) then an application that depends on this assumption may fail. One reason for the failure of the application may be that it uses significantly more resources than those used in the initial experiments that failed to detect the problem. See further discussion below. Another issue is that the probability that the application fails may indeed be negligible (as is the probability of detecting the failure in the initial experiments), but due to the importance of the application we are unwilling to tolerate even a negligible probability of failure.

Indeed, the fact that truthfulness is not reflected by computational (or even statistical) indistinguishability raises the question of motivation. The question is why should we care about a property that we cannot test. The answer is that others may be able to test this property and/or to benefit from its violation. The most evident settings in which this may occur are multi-party settings consisting of parties that have vastly different computational abilities (e.g., as underlying zero-knowledge interactive proof and perfectly binding commitment schemes (cf., [18, Chap. 4])). In such settings the indistinguishability requirement may refer to one (computationally-bounded) party, whereas truthfulness is essential for preventing events that refer to actions of the other party (which is not computationally-bounded). For example, a pseudorandom function is computationally indistinguishable from a random permutation, but if we face an opponent that is not computationally bounded and can illegitimately benefit from finding an image that has multiple preimages then we should insist on using a truthful implementation of a random permutation. The same considerations apply also if the opponent is computationally bounded but has extra information regarding the pseudorandom function (e.g., the opponent may know the succinct descryption of the function, hereafter called its seed). Note that in this example, computational indistinguishability may refer to one party (who does not know the function's seed), whereas truthfulness guarantees that the implemented function is a permutation and thus that the opponent (who may know the seed) cannot find an image that has multiple preimages.

Truthful implementations as an extension of complexity theory. Specializing our notion of a truthful implementation to the case of deterministic specifications yields the standard notion of efficient computation; that is, a truthful implementation of a function $f : \{0,1\}^* \to \{0,1\}^*$ is nothing but a polynomial-time algorithm for computing f. Similarly, an almost-truthful implementation of f is a probabilistic polynomial-time algorithm for computing f (with exponentially vanishing error probability). Thus, our notion of truthful implementations extends the standard study of polynomial-time computations from functions to probability distributions over functions (i.e., specifications).

1.3 Organization

In Section 2, we present formal definitions of the notions discussed above as well as basic observations regarding these notions. These are followed by a few known examples of non-trivial implementations of various random objects (which are retrospectively cast nicely in our formulation). In Section 3, we state a fair number of new implementations of various random objects, while deferring the constructions (and proofs) to subsequent corresponding sections (i.e., Sections 4 through 9). These implementations demonstrate the applicability of our notions to various domains such as functions, graphs, and codes. Indeed, Sections 4 through 9 are mutually independent and can be read in arbitrary order.

We call the reader's attention to Sections 10 and 11, which contain conclusions and open problems (Section 10) as well as a brief review of subsequent work (Section 11).

2 Formal Setting and General Observations

Throughout this work we let n denote the feasibility parameter. Specifically, feasible-sized objects have an explicit description of length poly(n), whereas huge objects have (explicit description) size exponential in n. The latter are described by functions from poly(n)-bit strings to poly(n)-bit strings. Whenever we talk of efficient procedures we mean algorithms running in poly(n)-time.

2.1 Specification

A huge random object is specified by a *computationally-unbounded* probabilistic Turing machine. For a fixed contents of the random-tape, such a machine defines a (possibly partial) function on the set of all binary strings. Such a function is called an instance of the specification. We consider the input-output relation of this machine when the random-tape is uniformly distributed. Loosely speaking, this is the random object specified by the machine.

For sake of simplicity, we confine our attention to machines that halt with probability 1 on every input. Furthermore, we will consider the input-output relation of such machines only on inputs of some specified length ℓ, where ℓ is always polynomially related to the feasibility parameter n. Thus, for such a probabilistic machine M and length parameter $\ell = \ell(n)$, with probability 1 over the choice of the random-tape for M, machine M halts on every $\ell(n)$-bit long input.

Definition 2.1 (specification): *For a fixed function $\ell : \mathbb{N} \to \mathbb{N}$, the* instance specified by a probabilistic machine *M,* random-tape *ω* and parameter *n is the function $M_{n,\omega}$ defined by letting $M_{n,\omega}(x)$ be the output of M on input $x \in \{0,1\}^{\ell(n)}$ when using the random-tape $\omega \in \{0,1\}^{\infty}$. The* random object specified by *M* and *n is defined as $M_{n,\omega}$ for a uniformly selected $\omega \in \{0,1\}^{\infty}$.*

Note that, with probability 1 over the choice of the random-tape, the random object (specified by M and n) depends only on a finite prefix of the random-tape. Let us clarify our formalism by casting in it several simple examples, which were considered before (cf. [20, 34]).

Example 2.2 (a random function): *A random function from n-bit strings to n-bit strings is specified by the machine M that, on input $x \in \{0,1\}^n$ (parameter n and random-tape ω), returns the* $\mathrm{idx}_n(x)$*-th n-bit block of ω, where* $\mathrm{idx}_n(x)$ *is the index of x within the set of n-bit long strings.*

Example 2.3 (a random permutation): *Let $N = 2^n$. A random permutation over $\{0,1\}^n \equiv [N]$ can be specified by uniformly selecting an integer $i \in [N!]$; that is, the machine uses its random-tape to determine $i \in [N!]$, and uses the i-th permutation according to some standard order. An alternative specification, which is easier to state (alas even more inefficient), is obtained by a machine that repeatedly inspect the N next n-bit strings on its random-tape, until encountering a run of N different values, using these as the permutation. Either way, once a permutation π over $\{0,1\}^n$ is determined, the machine answers the input $x \in \{0,1\}^n$ with the output $\pi(x)$.*

Example 2.4 (a random permutation coupled with its inverse): *In continuation to Example 2.3, we may consider a machine that selects π as before, and responds to input (σ, x) with $\pi(x)$ if $\sigma = 1$ and with $\pi^{-1}(x)$ otherwise. That is, the object specified here provides access to a random permutation as well as to its inverse.*

2.2 Implementations

Definition 2.1 places no restrictions on the complexity of the specification. Our aim, however, is to implement such specifications *efficiently*. We consider several types of implementations, where in all cases we aim at *efficient* implementations (i.e., machines that respond to each possible input within polynomial-time). Specifically, we consider two parameters:

1. The type of model used in the implementation. We will use either a polynomial-time *oracle machine having access to a random oracle* or a standard probabilistic polynomial-time machine (viewed as a deterministic *machine having access to a finite random-tape*).
2. The similarity of the implementation to the specification; that is, the implementation may be *perfect*, *statistically indistinguishable* or only *computationally indistinguishable* from the specification (by probabilistic polynomial-time oracle machines that try to distinguish the implementation from the specification by querying it at inputs of their choice).

Our real goal is to derive implementations by *ordinary machines* (having as good a quality as possible). We thus view implementations by *oracle machines having access to a random oracle* merely as a clean abstraction, which is useful in many cases (as indicated by Theorem 2.9 below).

Definition 2.5 (implementation by oracle machines): *For a fixed* (length) *function* $\ell : \mathsf{N} \to \mathsf{N}$*, a* (deterministic) *polynomial-time* oracle *machine* M *and oracle* f*, the* instance implemented by M^f and parameter n *is the function* M^f *defined by letting* $M^f(x)$ *be the output of* M *on input* $x \in \{0,1\}^{\ell(n)}$ *when using the oracle* f*. The* random object implemented by M with parameter n *is defined as* M^f *for a uniformly distributed* $f : \{0,1\}^* \to \{0,1\}$*.*

In fact, $M^f(x)$ depends only on the value of f on inputs of length bounded by a polynomial in $|x|$. Similarly, an ordinary probabilistic polynomial-time (as in the following definition) only uses a poly($|x|$)-bit long random-tape when invoked on input x. Thus, for feasibility parameter n, the machine handles $\ell(n)$-bit long inputs using a random-tape of length $\rho(n) = \text{poly}(\ell(n)) = \text{poly}(n)$, where (w.l.o.g.) ρ is 1-1.

Definition 2.6 (implementation by ordinary machines): *For fixed functions* $\ell, \rho : \mathsf{N} \to \mathsf{N}$*, an ordinary polynomial-time machine* M *and a string* r*, the* instance implemented by M and random-tape r *is the function* M_r *defined by letting* $M_r(x)$ *be the output of* M *on input* $x \in \{0,1\}^{\ell(\rho^{-1}(|r|))}$ *when using the random-tape* r*. The* random object implemented by M with parameter n *is defined as* M_r *for a uniformly distributed* $r \in \{0,1\}^{\rho(n)}$*.*

We stress that an instance of the implementation is fully determined by the machine M and the random-tape r (i.e., we disallow "implementations" that construct the object on-the-fly while depending on and keeping track of all previous queries and answers).[3]

For a machine M (either a specification or an implementation) we identify the pair (M, n) with the random object specified (or implemented) by machine M and feasibility parameter n.

Definition 2.7 (indistinguishability of the implementation from the specification): *Let* S *be a specification and* I *be an implementation, both with respect to the length function* $\ell : \mathsf{N} \to \mathsf{N}$*. We say that* I perfectly implements S *if, for every* n*, the random object* (I, n) *is distributed identically to the random object* (S, n)*. We say that* I closely-implements S *if, for every oracle machine* M *that on input* 1^n *makes at most* polynomially-many queries*, all of length* $\ell(n)$*, the following difference*

$$|\mathbf{Pr}[M^{(I,n)}(1^n) = 1] - \mathbf{Pr}[M^{(S,n)}(1^n) = 1]| \tag{1}$$

is negligible[4] *as a function of* n*. We say that* I pseudo-implements S *if the expression in* (1) *is negligible for every probabilistic polynomial-time oracle machine* M *that makes only queries of length equal to* $\ell(n)$*.*

[3]We mention that such "stateful implementations" were considered in the subsequent work of [7].

[4]A function $\mu : \mathsf{N} \to [0,1]$ is called negligible if for every positive polynomial p and all sufficiently large n's it holds that $\mu(n) < 1/p(n)$.

We stress that the notion of a close-implementation does not say that the objects (i.e., (I,n) and (S,n)) are statistically close; it merely says that they cannot be distinguished by a (computationally unbounded) machine that asks polynomially many queries. Indeed, the notion of pseudo-implementation refers to the notion of computational indistinguishability (cf. [24, 40]) as applied to functions (see [20]). Clearly, *any perfect implementation is a close-implementation*, and *any close-implementation is a pseudo-implementation.* Intuitively, the oracle machine M, which is sometimes called a (potential) distinguisher, represents a user that employs (or experiments with) the implementation. It is required that such a user cannot distinguish the implementation from the specification, provided that the user is limited in its access to the implementation or even in its computational resources (i.e., time).

Indeed, it is trivial to perfectly implement a random function (i.e., the specification given in Example 2.2) by using an *oracle machine* (with access to a random oracle). In contrast, the main result of Goldreich, Goldwasser and Micali [20] can be cast by saying that there exist a pseudo-implementation of a random function by an *ordinary machine*, provided that pseudorandom generators (or, equivalently, one-way function [6, 40, 25]) do exist. In fact, under the same assumption, it is easy to show that *every specification having a pseudo-implementation by an oracle machine also has a pseudo-implementation by an ordinary machine.* A stronger statement will be proven below (see Theorem 2.9).

Truthful implementations. An important notion regarding (non-perfect) implementations refers to the question of whether or not they satisfy properties that are enjoyed by the corresponding specification. Put in other words, the question is whether *each instance of the implementation is also an instance of the specification.* Whenever this condition holds, we call the implementation truthful. Indeed, every perfect implementation is truthful, but this is not necessarily the case for close-implementations. For example, a random function is a close-implementation of a random permutation (because it is unlikely to find a collision among polynomially-many pre-images); however, a random function is *not* a *truthful* implementation of a random permutation.

Definition 2.8 (truthful implementations): *Let S be a specification and I be an implementation. We say that I* is truthful to *S if for every n the support of the random object (I,n) is a subset of the support of the random object (S,n).*

Much of this work is focused on truthful implementations. The following simple result is very useful in the study of the latter. It asserts that we may focus on designing implementations by oracle machines (having access to a random oracle), and automatically obtain standard implementations (by ordinary probabilistic polynomial-time machines). We comment that this result is typically applied to (truthful) *close*-implementations by *oracle* machines, yielding (truthful) *pseudo*-implementations by *ordinary* machines.

Theorem 2.9 *Suppose that one-way functions exist. Then any specification that has a pseudo-implementation by an* oracle machine (*having access to a random oracle*) *also has a pseudo-implementation by an* ordinary machine. *Furthermore, if the former implementation is truthful then so is the latter.*

The sufficient condition is also necessary, because the existence of pseudorandom functions (i.e., a pseudo-implementation of a random function by an ordinary machine) implies the existence of one-way functions. In view of Theorem 2.9, whenever we seek truthful implementations (or, alternatively, whenever we do not care about truthfulness at all), we may focus on implementations by oracle machines.

Proof: First we replace the random oracle used by the former implementation by a pseudorandom oracle (i.e., aby a pseudorandom function that is available by the results of [20, 25]). Thus, no probabilistic polynomial-time distinguisher can detect the difference, except with negligible probability. Furthermore, the support of the pseudorandom oracle is a subset of the support of the random oracle, and so the truthful property is inherited by the latter implementation. Finally, we use an ordinary machine to emulate the oracle machine that has access to a pseudorandom oracle; that is, the ordinary machine selects a random seed for the pseudorandom function and emulates the computation of the original oracle machine with the corresponding oracle. ∎

Almost-Truthful implementations. Truthful implementations guarantee that each instance of the implementation is also an instance of the specification (and is thus "consistent with the specification"). A meaningful relaxation of this guarantee refers to the case that almost all the probability mass of the implementation is assigned to instances that are consistent with the specification (i.e., are in the support of the latter). Specifically, we refer to the following definition.

Definition 2.10 (almost-truthful implementations): *Let S be a specification and I be an implementation. We say that I* is almost-truthful to *S if the probability that (I, n) is not in the support of the random object (S, n) is bounded by a negligible function in n.*

Interestingly, almost-truthfulness is not preserved by the construction used in the proof of Theorem 2.9. In fact, there exists specifications that have almost-truthful close-implementations by oracle machines but not by ordinary machines (see Theorem 2.11 below). Thus, when studying almost-truthful implementations, one needs to deal directly with ordinary implementations (rather than focus on implementations by oracle-machines). Indeed, we will present a few examples of almost-truthful implementations that are not truthful.

Theorem 2.11 *There exists a specification that has an almost-truthful close-implementation by an oracle machine but has no almost-truthful implementation by an ordinary machine.*

We stress that the theorem holds regardless of whether or not the latter (almost-truthful) implementation is indistinguishable from the specification.

Proof: Consider the specification of a uniformly selected function $f : \{0,1\}^n \to \{0,1\}$ having (time-modified) Kolmogorov Complexity [32, Def. 7.5.1][5] greater than 2^{n-1}. That is, the specification machine scans its random-tape, looking for a block of 2^n bits of (time-modified) Kolmogorov Complexity greater than 2^{n-1}, and once found uses this block as a truth-table of the desired Boolean function. Since all but a negligible fraction of the functions have Kolmogorov Complexity greater than 2^{n-1}, an almost-truthful close-implementation *by an oracle machine* may just use a random function. On the other hand, any implementation *by an ordinary machine* (of randomness complexity ρ) induces a function $f : \{0,1\}^n \to \{0,1\}$ of (time-modified) Kolmogorov Complexity at most $(O(1) + \rho(n)) + \log_2(\mathrm{poly}(n) \cdot 2^n) = \mathrm{poly}(n)$. Thus, any such implementation yields a function that violates the specification, and so cannot even be "remotely" truthful. ■

2.3 Known non-trivial implementations

In view of Theorem 2.9, *when studying truthful implementations*, we focus on implementations by oracle machines. In these cases, we shorthand the phrase *implementation by an oracle machine* by the term *implementation*. Using the notion of truthfulness, we can cast the non-trivial implementation of a random permutation provided by Luby and Rackoff [34] as follows.

Theorem 2.12 [34]: *There exists a* truthful *close-implementation of the specification provided in Example 2.3. That is, there exists a truthful close-implementation of the specification that uniformly selects a permutation π over $\{0,1\}^n$ and responses to the query $x \in \{0,1\}^n$ with the value $\pi(x)$.*

Recall that Example 2.3 has a trivial close-implementation by a random function, but this trivial implementation is not truthful. The point of Theorem 2.12 is that it provides a *truthful* close-implementation of the specification provided in Example 2.3. On the other hand, even when ignoring the issue of truthfulness, it is non-trivial to provide a close-implementation of Example 2.4 (i.e., a random permutation along with its inverse).[6] However, Luby and Rackoff [34] have also provided a truthful close-implementation of Example 2.4.

[5] Loosely speaking, the (standard) Kolmogorov Complexity of a string s is the minimum length of a program Π that produce s. The time-modified Kolmogorov Complexity of a string s is the minimum, taken over programs Π that produce s, of $|\Pi| + \log_2(\mathtt{time}(\Pi))$, where $\mathtt{time}(\Pi)$ is the running-time of Π. We use time-modified Kolmogorov Complexity in order to allow for a recursive specification.

[6] A random function will fail here, because the distinguisher may distinguish it from a random permutation by asking for the inverse of a random image.

Theorem 2.13 [34]: *There exists a truthful close-implementation of the specification that uniformly selects a permutation π over $\{0,1\}^n$ and responses to the query $(\sigma, x) \in \{-1,+1\} \times \{0,1\}^n$ with the value $\pi^\sigma(x)$.*

Another known result that has the flavor of the questions that we explore was obtained by Naor and Reingold [38]. Loosely speaking, they provided a truthful close-implementation of a permutation selected uniformly among all permutations having a certain cycle-structure.

Theorem 2.14 [38]: *For any $N = 2^n$, $t = \text{poly}(n)$, and $C = \{(c_i, m_i) : i = 1, ..., t\}$ such that $\sum_{i=1}^{t} m_i c_i = N$, there exists a truthful close-implementation of a uniformly distributed permutation that has m_i cycles of size c_i, for $i = 1, ..., t$.*[7] *Furthermore, the implementation instance that uses the permutation π can also support queries of the form (x, j) to be answered by $\pi^j(x)$, for any $x \in \{0,1\}^n$ and any integer j (which is presented in binary).*

We stress that the latter queries are served in $\text{poly}(n)$-time also in the case that $j \gg \text{poly}(n)$.

2.4 A few general observations

Theorem 2.11 asserts the existence of specifications that cannot be implemented in an *almost-truthful* manner by an *ordinary* machine, regardless of the level of indistinguishability (of the implementation from the specification). We can get negative results that refer also to implementations by *oracle* machines, regardless of truthfulness, by requiring the implementation to be sufficiently indistinguishable (from the specification). Specifically:

Proposition 2.15 *The following refers to implementations by oracle machines and disregard the issue of truthfulness.*

1. *There exist specifications that cannot be closely-implemented.*
2. *Assuming the existence of one-way functions, there exist specifications that cannot be pseudo-implemented.*

The hypothesis in Part 2 can be relaxed: It suffices to assume the existence of NP-sets for which it is feasible to generate hard instances. For details see Appendix D.

Proof: Starting with Part 2, we note that the specification may be a deterministic process that invert a one-way function f (as in the hypothesis) at images of the user's choice (i.e., the query x is answered by the lexicographically first element in $f^{-1}(x)$). Certainly, this specification cannot be pseudo-implemented, because such an implementation would yield an algorithm that violates the hypothesis (of Part 2).[8] We may easily adapt this example such that the specification gives rise to a random object. For example, the specification may require that, given a pair of strings, one should use a random function to select one of these two strings, and answer with this string's inverse under the one-way function. A pseudo-implementation of this specification can also be shown to contradict the hypothesis. This establishes Part 2.

Turning to Part 1, we consider any fixed a function f that is computable in exponential-time but cannot be inverted, except for with negligible probability, by any polynomial-time machine that uses a random oracle. Such a function can be shown to exist by using a counting argument. The specification determines such a function, and inverts it at inputs of the user's choice. Observe that a close-implementation of such a function is required to successfully invert the function at random images, which is impossible (except for negligible probability, because the implementation is a polynomial-time machine (which uses a random oracle)). ■

[7] Special cases include involutions (i.e., permutations in which all cycles have length 2), and permutations consisting of a single cycle (of length N). These cases are cast by $C = \{(2, N/2)\}$ and $C = \{(N, 1)\}$, respectively. Note that this description presumes that C is fixed, which may be the case only when $N = 2^n$ is fixed. Thus, a proper formulation should either postulate that $C = C(N)$ is efficiently computable from N, or seek a "universal" object that answers queries in which C is given as part of the query.

[8] Consider the performance of the specification (resp., implementation) when queried on a randomly generated image, and note that the correctness of the answer can be efficiently verified. Thus, since the specification always inverts f on the given image, a pseudo-implementation must do the same (except with negligible probability), yielding a probabilistic polynomial-time algorithm that inverts f.

The randomness complexity of implementations: Looking at the proof of Theorem 2.9, it is evident that as far as pseudo-implementations by ordinary machines are concerned (and assuming the existence of one-way functions), *randomness can be reduced to any power of the feasibility parameter* (i.e., to n^ϵ for every $\epsilon > 0$). The same holds with respect to truthful pseudo-implementations. On the other hand, the proof of Theorem 2.11 suggests that this collapse in the randomness complexity cannot occur with respect to almost-truthful implementations by ordinary machines (regardless of the level of indistinguishability of the implementation from the specification).

Theorem 2.16 (a randomness hierarchy): *For every polynomial ρ, there exists a specification that has an almost-truthful close-implementation by an ordinary machine that uses a random-tape of length $\rho(n)$, but has no almost-truthful implementation by an ordinary machine that uses a random-tape of length $\rho(n) - \omega(\log n)$.*

Proof: Let $g(n) = \omega(\log n)$. Consider the specification that selects uniformly a string $r \in \{0,1\}^{\rho(n)}$ of (time-modified) Kolmogorov Complexity at least $\rho(n) - g(n)$, and responds to the query $i \in [2^n]$ with the $(1 + (i \bmod \rho(n)))$-th bit of r. Since all but an $\exp(-g(n)) = n^{-\omega(1)}$ fraction of the $\rho(n)$-bit long strings have such complexity, this specification is closely-implemented in an almost-truthful manner by a machine that uniformly selects $r \in \{0,1\}^{\rho(n)}$ (and responds as the specification). However, any implementation that uses a random-tape of length ρ', yields a function that assigns the first $\rho(n)$ arguments values that yield a $\rho(n)$-bit long string of (time-modified) Kolmogorov Complexity at most $(O(1) + \rho'(n)) + \log_2(\text{poly}(n)) = \rho'(n) + O(\log n)$. Thus, for $\rho'(n) = \rho(n) - 2g(n)$, the implementation cannot even be "remotely" truthful. ■

Composing implementations: A simple observation that is used in our work is that one can "compose implementations". That is, if we implement a random object R1 by an oracle machine that uses oracle calls to a random object R2, which in turn has an implementation by a machine of type T, then we actually obtain an implementation of R1 by a machine of type T. To state this result, we need to extend Definition 2.5 such that it applies to oracle machines that use (or rather have access to) arbitrary specifications (rather than a random oracle). Let us denote by $(M^{(S,n)}, n)$ an implementation by the oracle machine M (and feasibility parameter n) with oracle access to the specification (S, n), where we assume for simplicity that S uses the same feasibility parameter as M.

Theorem 2.17 *Let $Q \in \{\text{perfect}, \text{close}, \text{pseudo}\}$. Suppose that the specification (S_1, n) can be Q-implemented by $(M^{(S_2,n)}, n)$ and that (S_2, n) has a Q-implementation by an ordinary machine* (resp., by an oracle machine with a random oracle). *Then, (S_1, n) has a Q-implementation by an ordinary machine* (resp., by an oracle machine with a random oracle). *Furthermore, if both the implementations in the hypothesis are truthful* (resp., almost-truthful) *then so is the implementation in the conclusion.*

Proof: The idea is to simply replace (S_2, n) by its implementation, denoted (I_2, n), and thus obtain an implementation $(M^{(I_2,n)}, n)$ of (S_1, n). Next, by combining the machines M and I_2, we obtain a machine I of the same type as the type of machine I_2, and it holds that (I, n) yields a random object that is distributed identically to $(M^{(I_2,n)}, n)$. Thus, we obtain an implementation (I, n) of (S_1, n), Indeed, (I, n) inherits the truthfulness (resp., almost-truthfulness) of the two given implementations (i.e., $M^{(S_2,\cdot)}$ and I_2). Similarly, the analysis of the "quality" of the implementation (I, n) relies on the "quality" of the two given implementations. Details follow.

If both $M^{(I_2,\cdot)}$ and I_2 are perfect implementation of S_1 and S_2 respectively, then I is a perfect implementation of S_1. If the former are only close-implementations, then using the hypothesis that M is polynomial-time it follows that M only makes polynomially many queries to its oracle and thus invoking M a polynomial number of times results in a polynomial number of queries to its oracle. Using the second hypothesis (i.e., the "quality" of I_2), it follows that $M^{(I_2,n)}$ and $M^{(S_2,n)}$ are indistinguishable by polynomially many queries. Using the first hypothesis (i.e., the "quality" of $M^{(S_2,n)}$), it follows that $(I, n) \equiv (M^{(I_2,n)}, n)$ is a close-implementation of (S_1, n).

Lastly, let us spell out the argument for the case of pseudo-implementations, while using the term computationally-indistinguishable as shorthand for indistinguishable by probabilistic polynomial-time oracle machines. The first hypothesis asserts that $(M^{(S_2,n)}, n)$ and (S_1, n) are computationally-indistinguishable, and the second hypothesis asserts that (I_2, n) and (S_2, n) are computationally-indistinguishable. Our goal is to prove that $(M^{(I_2,n)}, n)$ and (S_1, n) are computationally-indistinguishable, which (by the first hypothesis) reduces to proving that $(M^{(I_2,n)}, n)$ and $(M^{(S_2,n)}, n)$ are computationally-indistinguishable. Now suppose, towards the contradiction, that some a probabilistic polynomial-time machine D distinguishes $(M^{(I_2,n)}, n)$ from $(M^{(S_2,n)}, n)$. Then, combining D and M, we obtain a machine that distinguishes (I_2, n) from (S_2, n), which contradicts the second hypothesis. The key point is that the fact that M is probabilistic polynomial-time (because it is an implementation machine), and so the combined distinguisher is also probabilistic polynomial-time (provided that so is D). ■

2.5 Objects of feasible size

In contrast to the rest of this work, in the current subsection we (shortly) discuss the complexity of generating random objects of feasible size (rather than huge random objects). In other words, we are talking about implementing a distribution on poly(n)-bit long strings, and doing so in poly(n)-time. This problem can be cast in our general formulation by considering specifications that ignore their input (i.e., have output that only depend on their random-tape). In other words, we may view objects of feasible size as constant functions, and consider a specification of such random objects as a distribution on constant functions. Thus, without loss of generality, the implementation may also ignore its input, and consequently in this case there is no difference between an implementation by ordinary machine and an implementation by oracle machine with a random oracle.

We note that perfect implementations of such distributions were considered before (e.g., in [1, 5, 17]), and distributions for which such implementations exist are called sampleable. In the current context, where the observer sees the entire object, the distinction between perfect implementation and close-implementation seems quite technical. What seems fundamentally different is the study of pseudo-implementations.

Theorem 2.18 *There exist specifications of feasible-sized objects that have no close-implementation, but do have* (both truthful and non-truthful) *pseudo-implementations.*

Proof: Any evasive pseudorandom distribution (cf. [21]) yields such a specification. Recall that a distribution is called evasive if it is infeasible to generate an element in its support (except with negligible probability), and is called pseudorandom if it is computationally indistinguishable from a uniform distribution on strings of the same length. It is known that evasive pseudorandom distributions do exist [21]. Note that, by definition, an evasive distribution has no close-implementation. On the other hand, any pseudorandom distribution can be pseudo-implemented by the uniform distribution (or any other pseudorandom distribution). Indeed, the latter implementation is not even almost-truthful with respect to the evasive pseudorandom distribution, because even a "remotely-truthful" implementation would violate the evasiveness condition. To allow also the presentation of a truthful implementation, we modify the specification such that with exponentially-small probability it produces some sampleable pseudorandom distribution, and otherwise it produces the evasive pseudorandom distribution. The desired truthful pseudo-implementation will always produce the former distribution (i.e., the sampleable pseudorandom distribution), and still the combined distribution has no close-implementation. ■

The proof of Theorem 2.18 (or rather the existence of evasive distributions) also establishes the existence of specifications (of feasible-sized objects) that have no truthful (and even no almost-truthful) implementation, regardless of the level of indistinguishability from the specification. Turning the table around, we ask whether there exist specifications of feasible-sized objects that have no pseudo-implementations, regardless of the truthfulness condition. A partial answer is provided by the following result, which relies on a non-standard assumption. Specifically, we assume the existence of a collision-resistant hash function; that is, a length-decreasing function $h : \{0,1\}^* \to \{0,1\}^*$ such that it is infeasible to form collisions under h (i.e., it is infeasible to find sufficiently long strings $x \neq y$ such that $f(x) = f(y)$).[9]

[9] We stress that the assumption used here (i.e., the existence of a *single* collision-resistant hash function) seems significantly stronger than the standard assumption that refers to the existence of an *ensemble* of collision-resistant functions (cf. [12] and [19, Def. 6.2.5]).

Proposition 2.19 *Assuming the existence of a collision-resistant hash* function, *there exists a specification of a random feasible-sized object that has no pseudo-implementation.*

Proof: The hypothesis implies the existence of a collision-resistant hash function h that shrinks its argument by exactly one bit (i.e., $|h(x)| = |x| - 1$).[10] Referring to this function h, consider the non-empty set $S_n \stackrel{\rm def}{=} \{(x,y) \in \{0,1\}^{n+n} : h(x) = h(y)\}$, and note that membership in $\cup_{n\in\mathbb{N}} S_n$ is easy to decide, while $\cup_{n\in\mathbb{N}} S_n$ is evasive. Consider the specification that consists of the uniform distribution over the set S_n, and note that this specification cannot be pseudo-implemented, because the likely event in which the implementation fails to hit S_n is easily detectable. ■

Open Problem 2.20 (stronger versions of Proposition 2.19): *Provide a specification of a random feasible-sized object that has no pseudo-implementation, while relying on some standard intractability assumption.*

Let us digress and consider close-implementations. For example, we note that Bach's elegant algorithm for generating random composite numbers along with their factorization [4] can be cast as a close-implementation of the said distribution.[11] We stress the highly non-trivial nature of the foregoing implementation (while recalling that it seems infeasible to find the factorization of a uniformly distributed composite number). A more elementary set of examples refers to the generation of integers (out of a huge domain) according to various "nice" distributions (e.g., the binomial distribution of N trials).[12] In fact, Knuth [29, Sec. 3.4.1] considers the generation of several such distributions, and his treatment (of integer-valued distributions) can be easily adapted to fit our formalism. This direction is further pursued in Appendix A. In general, recall that in the current context (where the observer sees the entire object), a close-implementation must be statistically close to the specification. Thus, almost-truthfulness follows "for free":

Proposition 2.21 *Any close-implementation of a specification of a feasible-sized object is almost-truthful to it.*

Multiple samples. Our general formulation can be used to specify an object that whenever invoked returns an independently drawn sample from the same distribution. Specifically, the specification may be by a machine that answers each "sample-query" by using a distinct portion of its random-tape (as coins used to sample from the basic distribution). Using a pseudorandom function, we may pseudo-implement multiple samples from any distribution for which one can pseudo-implement a single sample. That is:

Proposition 2.22 *Suppose that one-way functions exist, and let $D = \{D_n\}$ be a probability ensemble such that each D_n ranges over* $\mathrm{poly}(n)$*-bit long strings. If D can be pseudo-implemented then so can the specification that answers each query by an independently selected sample of D. Furthermore, the latter implementation is by an ordinary machine and is truthful provided that the former implementation is truthful.*

Proof: Consider first an implementation by an oracle machine that merely uses the random function to assign each query a random-tape to be used by the pseudo-implementation of (the single sample of the distribution) D. Since truthfulness and computational-indistinguishability are preserved by multiple (independent) samples (cf. [18, Sec. 3.2.3] for the latter), we are done as far as implementations by oracle machines are concerned. Using Theorem 2.9, the proposition follows. ■

[10] Given an arbitrary function h' as in the hypothesis, consider the function h'' defined by $h''(x) = h'(h'(x))$. Then, h'' is collision-resistant and $|h''(x)| \leq |x| - 2$. Now, consider the function h defined by $h(x) = h''(x)01^{|x|-|h''(x)|-2}$, and note that $|h(x)| = |x| - 1$ while h is also collision-resistant.

[11] We mention that Bach's motivation was to generate prime numbers P along with the factorization of $P - 1$, in order to allow efficient testing of whether a given number is a primitive element modulo P. Thus, one may say that Bach's paper provides a close-implementation (by an ordinary probabilistic polynomial-time machine) of the specification that selects at random an n-bit long prime P and answers the query g by 1 if and only if g is a primitive element modulo P. Note that the latter specification refers to a huge random object.

[12] That is, for a huge $N = 2^n$, we want to generate i with probability $p_i \stackrel{\rm def}{=} \binom{N}{i}/2^N$. Note $i \in \{0, 1, ...N\}$ has feasible size, and yet the problem is not trivial (because we cannot afford to compute all p_i's).

3 Our Main Results

We obtain several new implementations of random objects. All our implementations are either truthful or almost-truthful with respect to the corresponding specifications. We present the corresponding results in two categories referring to whether they yield truthful or only almost-truthful implementations.

3.1 Truthful Implementations

All implementations stated in this section are by (polynomial-time) *oracle machines* (which use a random oracle). Corresponding pseudo-implementations by ordinary (probabilistic polynomial-time) machines can be derived using Theorem 2.9. Namely, assuming the existence of one-way functions, *each of the specifications considered below can be pseudo-implemented in a truthful manner by an ordinary probabilistic polynomial-time machine.*

The basic technique underlying the following implementations is the embedding of additional structure that enables to efficiently answer the desired queries in a consistent way or to force a desired property. That is, this additional structure ensures truthfulness (with respect to the specification). The additional structure may cause the implementation to have a distribution that differs from that of the specification, but this difference is infeasible to detect (via the polynomially-many queries). In fact, the additional structure is typically randomized in order to make it undetectable, but each possible choice of coins for this randomization yields a "valid" structure (which in turn ensures truthfulness rather than only almost-truthfulness).

3.1.1 Supporting complex queries regarding Boolean functions

As mentioned above, a random Boolean function is trivially implemented (in a perfect way) by an oracle machine. By this we mean that the specification and the implementation merely serve the standard evaluation queries that refer to the values of a random function at various positions (i.e., query x is answered by the value of the function at x). Here we consider specifications that supports more powerful queries.

Example 3.1 (answering some parity queries regarding a random function): *Suppose that, for a random function $f : [2^n] \to \{0,1\}$, we wish to be able to provide the parity of the values of f on any desired interval of $[2^n]$. That is, we consider a specification defined by the machine that, on input (i,j) where $1 \leq i \leq j \leq 2^n$, replies with the parity of the bits residing in locations i through j of its random-tape. (Indeed, this specification refers to the length function $\ell(n) = 2n$.)*

Clearly, the implementation cannot afford to compute the parity of the corresponding values in its random oracle. Still, in Section 5 we present a *perfect* implementation of Example 3.1, as well as truthful close-implementations of more general types of random objects (i.e., answering any symmetric "interval" query). Specifically, we prove:

Theorem 3.2 (see Theorem 5.2)[13]: *For every polynomial-time computable function g, there exists a truthful close-implementation of the following specification of a random object. The specification machine uses its random-tape to define a random function $f : \{0,1\}^n \to \{0,1\}$, and answers the query $(\alpha, \beta) \in \{0,1\}^{n+n}$ by $g(\sum_{\alpha \leq s \leq \beta} f(s))$.*

3.1.2 Supporting complex queries regarding length-preserving functions

In Section 9 we consider specifications that, in addition to the standard evaluation queries, answer additional queries regarding a random length-preserving function. Such objects have potential applications in computational number theory, cryptography, and the analysis of algorithms (cf. [15]). Specifically, we prove:

[13] We mention that a related result was discovered before us by Naor and Reingold; see discussion at the end of Section 5. We also mention that, indeed, the current formulation of Theorem 5.2 follows from the special case in which g is the identity function.

Theorem 3.3 (see Theorem 9.2)[14]: *There exists a truthful close-implementation of the following specification. The specifying machine, uniformly selects a function $f : \{0,1\}^n \to \{0,1\}^n$, and, in addition to the standard evaluation queries, answers the inverse-query $y \in \{0,1\}^n$ with the set $f^{-1}(y)$.*

Alternatively, the implementation may answer with a uniformly distributed preimage of y under f (and with a special symbol in case no such preimage exists). A different type of queries is supported by the following implementation.

Theorem 3.4 (see Theorem 9.1): *There exists a truthful close-implementation of the following specification. The specifying machine, uniformly selects a function $f : \{0,1\}^n \to \{0,1\}^n$, and answers the query (x, m), where $x \in \{0,1\}^n$ and $m \in [2^{\text{poly}(n)}]$, with the value $f^m(x)$ (i.e., f iterated m times on x).*

This result is related to questions studied in [38, 39]; for more details, see Section 9.

3.1.3 Random graphs of various types

Random graphs have been extensively studied (cf. [8]), and in particular are known to have various properties. But does it mean that we can provide *truthful* close-implementations of uniformly distributed (huge) graphs having any of these properties?

Let us first consider a specification for a random N-vertex graph, where $N = 2^n$. Indeed, such a random graph can be specified by the machine, which viewing its random-tape ω as an N-by-N matrix (i.e., $\omega = (\omega_{i,j})_{i,j\in[N]}$), answers the input $(i,j) \in [N] \times [N]$ with the value $\omega_{i,j}$ if $i < j$, with the value $\omega_{j,i}$ if $i > j$. and with the value 0 otherwise (i.e., if $i = j$). Needless to say, this specification can be perfectly implemented (by a machine that uses its random oracle in an analogous manner). But how about implementing a uniformly distributed graph that has various properties?

Example 3.5 (uniformly distributed connected graphs): *Suppose that we want to implement a uniformly distributed* connected *graph (i.e., a graph uniformly selected among all connected N-vertex graph). An adequate specification may scan its random-tape, considering each N^2-bit long portion of it as a description of a graph, and answer adjacency-queries according to the first portion that yields a connected graph. Note that the specification works in time $\Omega(N^2)$, whereas an implementation needs to work in* $\text{poly}(\log N)$*-time. On the other hand, recall that a random graph is connected with overwhelmingly high probability. This suggests to implement a random connected graph by a random graph. Indeed, this yields a close-implementation,* but not a truthful one (*because occasionally, yet quite rarely, the implementation will yield an unconnected graph*).[15]

In Section 6 we present truthful close-implementations of Example 3.5 as well as of other (specifications of) uniformly distributed graphs having various additional properties. These are all special cases of the following general result:

Theorem 3.6 (see Theorem 6.2): *Let Π be a* monotone *graph property that is satisfied by a family of strongly-constructible* sparse *graphs. That is, for some negligible function μ (and every N), there exists a perfect implementation of a* (single) *N-vertex graph with $\mu(\log N) \cdot N^2$ edges that satisfies property* Π. *Then, there exists a truthful close-implementation of a uniformly distributed graph that satisfies property* Π.

We stress that Theorem 6.2 applies also to properties that are not satisfied (with high probability) by a random graph (e.g., having a clique of size $\sqrt{N}$). The proof of Theorem 6.2 relies on the following lemma, which may be of independent interest. Loosely speaking, the lemma asserts that if a monotone graph property Π is satisfied by some sparse graphs then a uniformly distributed graph having property Π is indistinguishable from a truly random graph.

[14] Note that in the specification it may happen (with negligible probability) that $|f^{-1}(y)| > \text{poly}(|y|)$, but this will never happen in the implementation.

[15] Note that failing to obtain a truthful implementation (by an oracle machine) does not allow us to derive (via Theorem 2.9) even an almost-truthful pseudo-implementation by an ordinary machine.

Lemma 3.7 (see Lemma 6.3): *Let* Π *be a* monotone *graph property that is satisfied by some* N*-vertex graph having* $\epsilon \cdot \binom{N}{2}$ *edges. Then, any machine that makes at most* q *adjacency queries to a graph, cannot distinguish a random* N*-vertex graph from a uniformly distributed* N*-vertex graph that satisfies* Π*, except than with probability* $O(q \cdot \sqrt{\epsilon}) + q \cdot N^{-(1-o(1))}$.

3.1.4 Supporting complex queries regarding random graphs

Suppose that we want to implement a random N-vertex graph along with supporting, in addition to the standard adjacency queries, also some complex queries that are hard to answer by only making adjacency queries. For example suppose that on query a vertex v, we need to provide a clique of size $\log_2 N$ containing v. In Section 7 we present a truthful close-implementations of this specification:

Theorem 3.8 (see Theorem 7.2): *There exists a truthful close-implementation of the following specification. The specifying machine selects uniformly an* N*-vertex graph and, in addition to the standard adjacency queries, answers* (Log-Clique) queries *of the form* v *by providing a random* $\lceil \log_2 N \rceil$*-vertex clique that contains* v *(and a special symbol if no such clique exists).*

Another result of a similar flavor refers to implementing a random graph while supporting additional queries that refer to a random Hamiltonian cycle in that graph.

Theorem 3.9 (see Theorem 7.3): *There exists a truthful close-implementation of the following specification. The specifying machine selects uniformly an* N*-vertex graph* G*, and in case* G *is Hamiltonian it uniformly selects a (directed) Hamiltonian cycle in* G*, which in turn defines a cyclic permutation* $\sigma : [N] \to [N]$*. In addition to the standard adjacency queries, the specification answers* travel queries *of the form* (trav, v, t) *by providing* $\sigma^t(v)$*, and* distance queries *of the form* (dist, v, w) *by providing the smallest* $t \geq 0$ *such that* $w = \sigma^t(v)$.

3.1.5 Random bounded-degree graphs of various types

Random bounded-degree graphs have also received considerable attention. In Section 8 we present truthful close-implementations of random bounded-degree graphs $G = ([N], E)$, where the machine specifying the graph answers the query $v \in [N]$ with the list of neighbors of vertex v. We stress that even implementing this specification is non-trivial if one insists on truthfully implementing *simple* random bounded-degree graphs (rather than graphs with self-loops and/or parallel edges). Furthermore, we present truthful close-implementations of random bounded-degree graphs having additional properties such as connectivity, Hamiltonicity, having logarithmic girth, etc. All these are special cases of the following result:

Theorem 3.10 (see Theorem 8.4:) *Let* $d > 2$ *be fixed and* Π *be a graph property that satisfies the following two conditions:*

1. *The probability that Property* Π *is not satisfied by a uniformly chosen* d*-regular* N*-vertex graph is negligible in* $\log N$.
2. *Property* Π *is satisfied by a family of strongly-constructible* d*-regular* N*-vertex graphs having girth* $\omega(\log\log N)$.

Then, there exists a truthful close-implementation of a uniformly distributed d*-regular* N*-vertex graph that satisfies property* Π.

The proof relies on the following lemma, which may be of independent interest. Loosely speaking, the lemma asserts that a random isomorphic copy of a fixed d-regular graph of large girth is indistinguishable from a truly random d-regular graph.

Lemma 3.11 (see Lemma 8.1): *For* $d > 2$*, let* $G = ([N], E)$ *be any* d*-regular* N*-vertex graph having girth* g*. Let* G' *be obtained by randomly permuting the vertices of* G *(and presenting the incidence lists in some canonical order). Then, any machine* M *that queries the graph for the neighborhoods of* q *vertices of its choice, cannot distinguish* G' *from a random* d*-regular* N*-vertex (simple) graph, except than with probability* $O(q^2/(d-1)^{(g-1)/2})$*. In the case that* $d = 2$ *and* $q < g-1$*, the probability bound can be improved to* $O(q^2/N)$.

3.2 Almost-Truthful Implementations

All implementations stated in this section are by *ordinary* (probabilistic polynomial-time) machines. All these results *assume the existence of one-way functions.*

Again, the basic technique is to embed a desirable structure, but (in contrast to Section 3.1) here the embedded structure forces the desired property only with very high probability. Consequently, the resulting implementation is only almost-truthful, which is the reason that we have to directly present implementations by ordinary machines.

A specific technique that we use is obtaining a function by taking a value-by-value combination of a pseudorandom function and a function of a desired combinatorial structure. The combination is done such that the combined function inherits both the pseudorandomness of the first function and the combinatorial structure of the second function (in analogy to a construction in [26]). In some cases, the combination is by a value-by-value XOR, but in others it is by a value-by-value OR with a second function that is very sparse.

3.2.1 Random codes of large distance

In continuation to the discussion in the introduction, we prove:

Theorem 3.12 (see Theorem 4.2): *For $\delta = 1/6$ and $\rho = 1/9$, assuming the existence of one-way functions, there exists an almost-truthful pseudo-implementation of the following specification: The specification machine uses its random-tape to uniformly select a code $C \subset \{0,1\}^n$ having cardinality $K \stackrel{\text{def}}{=} 2^{\rho n}$ and distance at least δn, and answers the query $i \in [K]$ with the i-th element in C.*

We comment that the above description actually specifies (and implements) an encoding algorithm for the corresponding code. It would be very interesting if one can also implement a corresponding decoding algorithm; see further discussion in Section 4.

3.2.2 Random graphs of various types

Having failed to provide truthful pseudo-implementations to the following specifications, we provide almost-truthful ones.

Theorem 3.13 (see Theorem 6.6): *Let $c(N) = (2-o(1))\log_2 N$ be the largest integer i such that the expected number of cliques of size i in a random N-vertex graph is larger than one. Assuming the existence of one-way functions, there exist almost-truthful pseudo-implementations of the following specifications:*

1. A random graph of Max-Clique $c(N) \pm 1$: *The specification uniformly selects an N-vertex graph having maximum clique size $c(N) \pm 1$, and answers edge-queries accordingly.*
2. A random graph of Chromatic Number $(1 \pm o(1)) \cdot N/c(N)$: *The specification uniformly selects an N-vertex graph having Chromatic Number $(1 \pm \log_2^{-1/3} N) \cdot N/c(N)$, and answers edge-queries accordingly.*

We mention that Theorem 6.7 provides an almost-truthful pseudo-implementation of a specification that refers to a uniformly distributed graph that satisfies both the aforementioned properties as well as several other famous properties that are satisfied (w.h.p.) by random graphs. Thus, this implementation not only looks as a random graph but rather satisfies all these properties of a random graph (although determining whether a huge graph satisfies any of these properties is infeasible).

One property of random graphs that was left out of Theorem 6.7 is having high (global) connectivity property. That is, we seek an almost-truthful pseudo-implementation of a uniformly distributed graph having a high global connectivity property (i.e., each pair of vertices is connected by many vertex-disjoint paths). Failing to provide such an implementation, we provide instead an almost-truthful pseudo-implementation of a random graph for which almost all pairs of vertices enjoy a high connectivity property.

Theorem 3.14 (see Theorem 6.8): *For every positive polynomial p, assuming the existence of one-way functions, there exists an almost-truthful pseudo-implementation of the following specification. The specifying machine selects a graph that is uniformly distributed among all N-vertex graphs for which all but at most an $\epsilon(N) \stackrel{\text{def}}{=} 1/p(\log_2 N)$ fraction of the vertex pairs are connected by at least $(1-\epsilon(N)) \cdot N/2$ vertex-disjoint paths. Edge-queries are answered accordingly.*

Interestingly, the same implementation works for all polynomials p; that is, the implementation is independent of p, which is only used for defining the specification. We note that a subsequent work by Alon and Nussboim [3] provides the implementation we failed to achieve (i.e., an almost-truthful pseudo-implementation of a uniformly distributed graph having a high global connectivity property).

4 Implementing Random Codes of Large Distance

For sufficiently small $\rho, \delta > 0$, we consider codes having relative rate ρ and relative distance δ; that is, we consider subsets $C \subset \{0,1\}^n$ such that $|C| = 2^{\rho n}$ and every two distinct codewords (i.e., $\alpha, \beta \in C$) disagree on at least δn coordinates. Such a code is called good. A random set of $K \stackrel{\text{def}}{=} 2^{\rho n}$ strings of length n is good with overwhelmingly high probability. Thus, for a random function $f : [K] \to \{0,1\}^n$, setting $C = \{f(i) : i \in [K]\}$ yields an almost-truthful close-implementation of a random code that is good, where the specification is required to answer the query i with the i-th codeword (i.e., the i-th element in the code). Recall that it is not clear what happens when we replace f by a pseudorandom function (i.e., it may be the case that the resulting code has very small distance, although most pairs of codewords are definitely far apart). Thus, in order to get an almost-truthful pseudo-implementation (by ordinary probabilistic polynomial-time machines) we use a different approach.

Construction 4.1 (implementing a good random code): *For $k = \rho n$, we select a random k-by-n matrix M, and consider the linear code generated by M (i.e., the codewords are obtained by all possible linear combinations of the rows of M). Now, using a pseudorandom function $f_s : \{0,1\}^k \to \{0,1\}^n$, where $s \in \{0,1\}^n$, we consider the code $C_{M,s} = \{f_s(v) \oplus vM : v \in \{0,1\}^k\}$. That is, our implementation uses the random-tape (M,s), and provides the i-th codeword of the code $C_{M,s}$ by returning $f_s(i) \oplus iM$, where $i \in [2^k]$ is viewed as a k-dimensional row vector (or a k-bit long string).*

To see that Construction 4.1 is a pseudo-implementation of a random code, consider what happens when the pseudorandom function is replaced by a truly random one (in which case we may ignore the nice properties of the random linear code generated by M).[16] Specifically, for any matrix M and any function $f : [K] \to \{0,1\}^n$, we consider the code $C_M^f = \{f(v) \oplus vM : v \in \{0,1\}^k\}$. Now, for any fixed choice of M and a truly random function $\phi : [K] \to \{0,1\}^n$, the code C_M^ϕ is a random code. Thus, the pseudorandomness of the function ensemble $\{f_s\}_{s \in \{0,1\}^n}$ implies that, for a uniformly chosen $s \in \{0,1\}^n$, the code $C_{M,s} = C_M^{f_s}$ is computationally indistinguishable from a random code. The reason being that ability to distinguish selected codewords of $C_M^{f_s}$ (for a random $s \in \{0,1\}^n$) from codewords of C_M^ϕ (for a truly random function $\phi : [K] \to \{0,1\}^n$) yields ability to distinguish the corresponding f_s from ϕ.

To see that Construction 4.1 is almost-truthful to the good code property, fix any (pseudorandom) function f and consider the code $C_M = \{f(v) \oplus vM : v \in \{0,1\}^k\}$, when M is a random k-by-n matrix. Fixing any pair of distinct strings $v, w \in \{0,1\}^k$, we show that with probability at least 2^{-3k} (over the possible choices of M), the codewords $f(v) \oplus vM$ and $f(w) \oplus wM$ are at distance at least δn, and it follows that with probability at least $1 - 2^{-k}$ the code C_M has a distance at least δn. Thus, for a random M, we consider the Hamming weight of $(f(v) \oplus vM) \oplus (f(w) \oplus wM)$, which in turn equals the Hamming weight of $r \oplus uM$, where $r = f(v) \oplus f(w)$ and $u = v \oplus w$ are fixed. The weight of $r \oplus uM$ behaves as a binomial distribution (with success probability 1/2), and thus the probability that the weight is less than δn is upper-bounded by $2^{-(1-\mathbf{H}_2(\delta)) \cdot n}$, where $\mathbf{H}_2$ denotes the binary entropy function. So we need $1 - \mathbf{H}_2(\delta) \cdot n > 3k$ to holds, and indeed it does hold for appropriate choices of δ and ρ (e.g, $\delta = 1/6$ and $\rho = 1/9$). Specifically, recalling that $k = \rho n$, we need $1 - \mathbf{H}_2(\delta) > 3\rho$ to hold. We get:

[16] In particular, note that the resulting code is unlikely to be linear. Furthermore, any $n - O(1) > k$ codewords are likely to be linearly independent (both when we use a random function or a pseudorandom one).

Theorem 4.2 *For any $\delta \in (0,1/2)$ and $\rho \in (0, 1-\mathbf{H}_2(\delta))/3$, assuming the existence of one-way functions, there exists an almost-truthful pseudo-implementation by an ordinary machine of the following specification: The specification machine uses its random-tape to uniformly select a code $C \subset \{0,1\}^n$ having cardinality $K \stackrel{\text{def}}{=} 2^{\rho n}$ and distance at least δn, and answers the query $i \in [K]$ with the i-th element in C.*

We comment that Construction 4.1 actually implements an encoding algorithm for the corresponding code, which is actually what is required in the specification. It would be very interesting if one could also implement a corresponding decoding algorithm. Note that the real challenge is to achieve "decoding with errors" (i.e., decode corrupted codewords rather than only decode uncorrupted codewords).[17] Specifically,

Open Problem 4.3 (implementing encoding and decoding for a good random code): *Provide an almost-truthful pseudo-implementation, even by an oracle machine, to the following specification. For some $\delta \in (0,1/2)$ and $\rho \in (0, \Omega(1-\mathbf{H}_2(\delta)))$, the specification machine selects a code $C \subset \{0,1\}^n$ as in Theorem 4.2, and answers queries of two types:*

Encoding queries: *For $i \in [K]$, the query (enc, i) is answered with the i-th element in C.*

Decoding queries: *For very $w \in \{0,1\}^n$ that is at distance at most $\delta n/3$ from C, the query (dec, w) is answered by the index of the* (unique) *codeword that is closest to w.*

Indeed, we are interested in an implementation by an ordinary machine, but as stated in Section 10, it may make sense to first consider implementations by oracle machines. Furthermore, it would be nice to obtain truthful implementations, rather than almost-truthful ones. In fact, it will even be interesting to have a truthful pseudo-implementation of the specification stated in Theorem 4.2.

5 Boolean Functions and Interval-Sum Queries

In this section we show that the specification of Example 3.1 can be perfectly implemented (by an oracle machine). Recall that we seek to implement access to a random function $f : \{0,1\}^n \to \{0,1\}$ augmented with answers regarding the parity (or XOR) of the values of f on given intervals, where the intervals are with respect to the standard lex-order of n-bit string. That is, the query $q = (\alpha, \beta) \in \{0,1\}^{n+n}$, where $0^n \leq \alpha \leq \beta \leq 1^n$, is to be answered by $\oplus_{\alpha \leq s \leq \beta} f(s)$. The specification can answer this query in the straightforward manner, but an implementation cannot afford to do so (because a straightforward computation may take $2^n = 2^{|q|/2}$ steps). Thus, the implementation will do something completely different.[18]

We present an oracle machine that uses a random function $f' : \cup_{i=0}^{n} \{0,1\}^i \to \{0,1\}$. Using f', we define $f : \{0,1\}^n \to \{0,1\}$ as follows. We consider a binary tree of depth n and associate its i^{th} level vertices with strings of length i such that the vertex associated with the string s has a left (resp., right) child associated with the string $s0$ (resp., $s1$). As a mental experiment, going from the root to the leaves, we label the tree's vertices as follows:

1. We label the root (i.e., the level-zero vertex, which is associated with λ) by the value $f'(\lambda)$.
2. For $i = 0, ..., n-1$, and each internal vertex v at level i, we label its *left* child by the value $f'(v0)$, and label its *right* child by the XOR of the label of v and the value $f'(v0)$.

   (Thus, the label of v equals the XOR of the values of its children.)

[17] Note that a simple modification of Construction 4.1 (e.g., replacing the i-th codeword, w, by the new codeword (i, w)), allows trivial decoding of uncorrupted codewords.

[18] The following implementation is not the simplest one possible, but we chose to present it because it generalizes to yield a proof of Theorem 5.2 (i.e., interval-sum rather than interval-sum-mod-2). A simpler implementation of Example 3.1, which does not seem to generalize to the case of interval-sum (as in Theorem 5.2), was suggested to us by Phil Klein, Silvio Micali, and Dan Spielman. The idea is to reduce the problem of Example 3.1 to the special case where we only need to serve interval-queries for intervals starting at 0^n; that is, we only need to serve (interval) queries of the form $(0^n, \beta)$. (Indeed, the answer to a query (α', β'), where $\alpha' \neq 0^n$, can be obtained from the answers to the queries $(0^n, \alpha'')$ and $(0^n, \beta')$, where α'' is the string preceding α'. Next observe that the query $(0^n, \beta)$ can be served by $f'(\beta)$, where $f' : \{0,1\}^n \to \{0,1\}$ is a random function (given as oracle).

3. The value of f at $\alpha \in \{0,1\}^n$ is defined as the label of the leaf associated with α.

By using induction on $i = 0, ..., n$, it can be shown that the level i vertices are assigned uniformly and independently distributed labels (which do depend, of course, on the level $i-1$ labels). Thus, f is a random function. Furthermore, the label of each internal node v equals the XOR of the values of f on all leaves in the subtree rooted at v.

Note that the random function f' is used to directly assign (random) labels to all the left-siblings. The other labels (i.e., of right-siblings) are determined by XORing the labels of the parent and the left-sibling. Furthermore, the label of each node in the tree is determined by XORing at most $n+1$ values of f' (residing in appropriate left-siblings). Specifically, the label of the vertex associated with $\sigma_1 \cdots \sigma_i$ is determined by the f'-values of the strings $\lambda, 0, \sigma_1 0, ..., \sigma_1 \cdots \sigma_{i-1} 0$. Actually, the label of the vertex associated with $\alpha 1^j$, where $\alpha \in \{\lambda\} \cup \{0,1\}^{|\alpha|-1}0$ and $j \geq 0$, is determined by the f'-values of $j+1$ vertices (i.e., those associated with $\alpha, \alpha 0, \alpha 10 ..., \alpha 1^{j-1} 0$).

$$
\begin{aligned}
\text{label}(\alpha 1^j) &= \text{label}(\alpha 1^{j-1}) \oplus \text{label}(\alpha 1^{j-1} 0) \\
&\vdots \\
&= \text{label}(\alpha) \oplus \text{label}(\alpha 0) \cdots \oplus \text{label}(\alpha 1^{j-2} 0) \oplus \text{label}(\alpha 1^{j-1} 0) \\
&= f'(\alpha) \oplus f'(\alpha 0) \cdots \oplus f'(\alpha 1^{j-2} 0) \oplus f'(\alpha 1^{j-1} 0)
\end{aligned}
$$

Thus, we obtain the value of f at any n-bit long string by making at most $n+1$ queries to f'. More generally, we can obtain the label assigned to each vertex by making at most $n+1$ queries to f'. It follows that we can obtain the value of $\oplus_{\alpha \leq s \leq \beta} f(s)$ by making $O(n^2)$ queries to f'. Specifically, the desired value is the XOR of the leaves residing in at most $2n-1$ full binary sub-trees, and so we merely need to XOR the labels assigned to the roots of these sub-trees. Actually, $O(n)$ queries can be shown to suffice, by taking advantage on the fact that we need not retrieve the labels assigned to $O(n)$ arbitrary vertices (but rather to vertices that correspond to roots of sub-trees with consecutive leaves). We get:

Theorem 5.1 *There exists a perfect implementation (by an oracle machine) of the specification of Example 3.1.*

The foregoing procedure can be generalize to handle queries regarding any (efficiently computable) symmetric function of the values assigned by f to any given interval. In fact, it suffices to answer queries regarding the sum of these values. We thus state the following result.

Theorem 5.2 *There exists a truthful close-implementation (by an oracle machine) of the following specification of a random object. The specification machine uses its random-tape to define a random function $f : \{0,1\}^n \to \{0,1\}$, and answers the query $(\alpha, \beta) \in \{0,1\}^{n+n}$ by $\sum_{\alpha \leq s \leq \beta} f(s)$.*

Note that, unlike in the case of Theorem 5.1, the implementation is not perfect, which is the reason that we explicitly mention that it is truthful.

Proof: All that is needed in order to extend the "XOR construction" is to make sure that the label of each vertex v equals the sum (rather than the sum mod 2) of the labels of all the leaves in the sub-tree rooted at v. In particular, internal nodes should be assigned random labels according to the binomial distribution, which makes the implementation more complex (even for assigning labels to the root and more so for assigning labels to left-siblings after their parents was assigned a label). Let us start with an overview:

1. We label the root by a value generated according to the binomial distribution; that is, the root (of the depth-n binary tree) is assigned the value j with probability $\binom{N}{j}/2^N$, where $N \stackrel{\text{def}}{=} 2^n$. This random assignment will be implemented using the value $f'(\lambda)$, where here f' is a random function ranging over poly(n)-bit long strings rather than over a single bit (i.e., $f' : \cup_{i=0}^{n} \{0,1\}^i \to \{0,1\}^{\text{poly}(n)}$).

2. For $i = 0, ..., n-1$, and each internal vertex v at level i, we label its *left* child as follows, by using the value $f'(v0)$. Suppose that v is assigned the value $T \leq 2^{n-i}$. We need to select a random pair of integers (l, r) such that $l + r = T$ and $0 \leq l, r \leq 2^{n-i-1}$. Such a pair should be selected with probability that equals the probability that, conditioned on $l + r = T$, the pair (l, r) is selected when l and r are distributed according to the binomial distribution (of 2^{n-i-1} trials). That is, let $M = 2^{n-i}$ be the number of leaves in the tree rooted at v. Then, for $l + r = T$ and $0 \leq l, r \leq M/2$, the pair (l, r) should be selected with probability $\binom{M/2}{l} \cdot \binom{M/2}{r} / \binom{M}{l+r}$.
3. As before, the value of f at $\alpha \in \{0,1\}^n$ equals the label of the leaf associated with α.

Of course, the above two types of sampling procedures have to be implemented in poly(n)-time, rather than in poly(2^n)-time (and poly$(n2^{n-i})$-time, respectively). These implementations cannot be perfect (because some of the events occur with probability $2^{-N} = 2^{-2^n}$), but it suffices to provide implementations that generates these samples with approximately the right distribution (e.g., with deviation at most 2^{-n} or so). The details concerning these implementations are provided in an Appendix A.

We stress that the sample (or label) generated for the (left sibling) vertex associated with $\alpha = \alpha'0$ is produced based on the randomness provided by $f'(\alpha)$. However, the actual sample (or label) generated for this vertex depends also on the label assigned to its parent. (Indeed, this is different from the case of XOR.) Thus, to determine the label assigned to any vertex in the tree, we need to obtain the labels of all its ancestors (up-to the root). Specifically, let $S_1(N, \rho)$ denote the value sampled from the binomial distribution (on N trials), when the sampling algorithm uses coins ρ; and let $S_2(M, T, \rho)$ denote the value assigned to the left-child, when its parent (which is the root of a tree with M leaves) is assigned the value T, and the sampling algorithm uses coins ρ. Then, the label of the vertex associated with $\alpha = \sigma_1 \cdots \sigma_t$, denoted label$(\alpha)$, is obtained by computing the labels of all its ancestors as follows. First, we compute label$(\lambda) \leftarrow S_1(N, f'(\lambda))$. Next, for $i = 1, ..., t$, we obtain label$(\sigma_1 \cdots \sigma_i)$ by computing

$$\mathrm{label}(\sigma_1 \cdots \sigma_{i-1}0) \leftarrow S_2(2^{n-(i-1)}, \mathrm{label}(\sigma_1 \cdots \sigma_{i-1}), f'(\sigma_1 \cdots \sigma_{i-1}0)),$$

and if necessary (i.e., $\sigma_i = 1$) by computing

$$\mathrm{label}(\sigma_1 \cdots \sigma_{i-1}1) \leftarrow \mathrm{label}(\sigma_1 \cdots \sigma_{i-1}) - \mathrm{label}(\sigma_1 \cdots \sigma_{i-1}0).$$

That is, we first determine the label of the root (using the value of f' at λ); and next, going along the path from the root to α, we determine the label of each vertex based on the label of its parent (and the value of f' at the left-child of this parent). Thus, the computation of the label of α, only requires the value of f' on $|\alpha| + 1$ strings. As in the case of XOR, this allows to answer queries (regarding the sum of the f-values in intervals) based on the labels of $O(n)$ internal nodes, where each of these labels depend only on the value of f' at $O(n)$ points. (In fact, as in the case of XOR, one may show that the values of these related internal nodes depend only on the value of f' at $O(n)$ points.)

Regarding the quality of the implementation, by the above description it is clear that the label of each internal node equals the sum of the labels of its children, and thus the implementation is truthful. To analyze its deviation from the specification, we consider the *mental experiment* in which both sampling procedures are implemented perfectly (rather than almost so), and show that in such a case the resulting implementation is perfect. Specifically, using induction on $i = 0, ..., n$, it can be shown that the level i vertices are assigned labels that are independently distributed, where each label is distributed as the binomial distribution of 2^{n-i} trials. (Indeed, the labels assigned to the vertices of level i do depend on the labels assigned in level $i-1$.) Thus, if the deviation of the actual sampling procedures is bounded by $2^{-n} \cdot \epsilon$, then the actual implementation is at statistical distance at most ϵ from the specification.[19] The latter statement is actually stronger than required for establishing the theorem. ■

Open problems: Theorem 5.2 provides a truthful implementation for any (feasibly-computable) symmetric function of the values assigned by a random function over any interval of $[N] \equiv \{0,1\}^n$. Two natural extensions are suggested below.

[19] We can afford to set $\epsilon = \exp(-\mathrm{poly}(n)) < 1/\mathrm{poly}(N)$, because the running time of the actual sampling procedures is poly-logarithmic in the desired deviation.

Open Problem 5.3 (non-symmetric queries): *Provide a truthful close-implementation to the following specification. The specification machine defines a random function $f : \{0,1\}^n \to \{0,1\}$, and answers queries of the form $(\alpha,\beta) \in \{0,1\}^{n+n}$ with the value $g(f(\alpha),...,f(\beta))$, where g is some simple function. For example, consider $g(\sigma_1,...,\sigma_t)$ that returns the smallest $i \in [t]$ such that $\sigma_i \cdots \sigma_{i+\lfloor 1+\log_2 t\rfloor -1} = 1^{1+\lfloor \log_2 t\rfloor}$ (and a special symbol if no such i exists). More generally, consider a specification machine that answers queries of the form $(k,(\alpha,\beta))$ by returning smallest $i \in [t]$ such that $\sigma_i \cdots \sigma_{i+k-1} = 1^k$, where σ_j is the j-th element in the sequence $(f(\alpha),...,f(\beta))$.*

Note that the latter specification is interesting mostly for $k \in \{\omega(\log n),...,n+\omega(\log n)\}$. For $k \le k_{\rm sm} = O(\log n)$ we may just make sure (in the implementation) that any consecutive interval of length $2^{k_{\rm sm}}n^2$ contains a run of $k_{\rm sm}$ ones.[20] Once this is done, queries (referring to $k \le k_{\rm sm}$) may be served (by the implementation) in a straightforward way (i.e., by scanning at most two such consecutive intervals, which in turn contain $2^{k_{\rm sm}+1}n^2 = \mathrm{poly}(n)$ values). Similarly, for $k \ge k_{\rm lg} = n + \omega(\log n)$, we may just make sure (in the implementation) that no pair of consecutive intervals, each of length $5n$, has a run of $\min(k_{\rm lg}, 2n)$ ones.

Open Problem 5.4 (beyond interval queries): *Provide a truthful close-implementation to 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 $\oplus_{\alpha\in S} f(\alpha)$. In Example 3.1 the class of sets is all intervals of $[N] \equiv \{0,1\}^n$, represented by their pair of end-points. Another natural case is the class of sub-cubes of $\{0,1\}^n$; that is, a set S is specified by an n-sequence over $\{0,1,*\}$ such that the set specified by the sequence $(\sigma_1,...,\sigma_n)$ contains the n-bit long string $\alpha_1 \cdots \alpha_n$ if and only if $\alpha_i = \sigma_i$ for every $\sigma_i \in \{0,1\}$.*

In both cases (i.e., Problems 5.3 and 5.4), even if we do not require truthfulness, a pseudo-implementation may need to be "somewhat truthful" anyhow (i.e., if the implementation answers the compound queries in a non-consistent manner then it may be distinguished from the specification because a distinguisher may check consistency). At least, a potential implementation seems to be in trouble if it "lies bluntly" (e.g., answers each query by an independent random bit).

An application to streaming algorithms: Motivated by a computational problem regarding massive data streams, Feigenbaum *et. al.* [14] considered the problem of constructing a sequence of N random variables, $X_1,...,X_N$, over $\{\pm 1\}$ such that

1. The sequence is "range-summable" in the sense that given $t \in [N]$ the sum $\sum_{i=1}^{t} X_i$ can be computed in $\mathrm{poly}(\log N)$-time.
2. The random variables are almost 4-wise independent (in a certain technical sense).

Using the techniques underlying Theorem 5.2, for any $k \le \mathrm{poly}(\log N)$ (and in particular for $k = 4$), we can construct a sequence that satisfies the above properties. In fact, we get a sequence that is almost k-wise independent in a stronger sense than stated in [14] (i.e., we get a sequence that is statistically close to being k-wise independent).[21] This is achieved by using the construction presented in the proof of Theorem 5.2, except that f' is a function selected uniformly from a family of $k\cdot(n+1)$-wise independent functions rather than being a truly random function, where $n = \log_2 N$ (as above). Specifically, we use functions that map $\{0,1\}^{n+1} \equiv \cup_{i=0}^{n}\{0,1\}^i$ to $\{0,1\}^{\mathrm{poly}(n)}$ in a $k\cdot(n+1)$-wise independent manner, and recall that such functions can be specified by $\mathrm{poly}(n)$ many bits and evaluated in $\mathrm{poly}(n)$-time (since $k \le \mathrm{poly}(n)$). In the analysis, we use the fact that the values assigned by f' to vertices in each of the $(n+1)$ levels of the tree are k-wise independent. Thus, we can prove by induction on $i = 0,...,n$, that every k vertices at level i are assigned labels according to the correct distribution (up to a small deviation). Recall that, as stated in Footnote 19, we can obtain statistical deviation that is negligible in N (in this case, with respect to a k-wise independent sequence).

[20] That is, the random function $f : [N] \to \{0,1\}$ is modified such that, for every $j \in [N/2^{k_{\rm sm}}n^2]$, the interval $[(j-1)2^{k_{\rm sm}}n^2+1,...,j2^{k_{\rm sm}}n^2]$ contains a run of $k_{\rm sm}$ ones. This modification can be performed on-the-fly by scanning the relevant interval and setting to 1 a random block of $k_{\rm sm}$ locations if necessary. Note that, with overwhelmingly high probability, no interval is actually modified.

[21] This construction was actually discovered before us by Naor and Reingold (cf. [16, Lem. 2]); see further discussion at the end of this section.

A historical note: As noted above, the ideas underlying the proof of Theorem 5.2 were discovered by Moni Naor and Omer Reingold (as early as in 1999). Actually, their construction was presented within the framework of limited independence (i.e., as in the former paragraph), rather than in the framework of random functions (used throughout this section). In fact, Naor and Reingold came-up with their construction in response to a question raised by the authors of [14] (but their solution was not incorporated in [14]). The Naor–Reingold construction was used in the subsequent work of [16] (see [16, Lem. 2]).

6 Random Graphs Satisfying Global Properties

Suppose that you want to run some simulations on huge random graphs. You actually take it for granted that the random graph is going to be Hamiltonian, because you are willing to discard the negligible probability that a random graph is not Hamiltonian. Suppose that you want to be able to keep succinct representations of these graphs and/or that you want to generate them using few random bits. A natural idea that comes to mind is using pseudorandom functions [20] in order to efficiently generate and store representations of these graphs. *But are the graphs generated this way really Hamiltonian?*

The point is that being Hamiltonian is a global property of the graph, which in turn is a huge (i.e., $\exp(n)$-sized) object. This global property cannot be decided by checking the adjacency of polynomially many (i.e., $\text{poly}(n)$-many) vertex-pairs, and so its violation cannot be translated to a contradiction of the pseudorandomness of the function. Indeed, the substitution of a random function (or a random graph) by a pseudorandom one is not guaranteed to preserve the global property. Specifically, it may be the case that all pseudorandom graphs are even disconnected.[22] So, *can we efficiently generate huge Hamiltonian graphs?* As we show below, the answer to this question is positive.

In this section we consider the implementation of various types of huge random graphs. We stress that we refer to simple and labeled graphs; that is, we consider graphs without self-loops or parallel edges, and with labeled vertices (i.e., the 3-vertex graph consisting of the edge $(1,2)$ is different from the 3-vertex graph consisting of the edge $(1,3)$). In this section, implementing a graph means answering adjacency queries; that is, the answer to the query (u,v) should indicate whether or not u and v are adjacent in the graph. Recall that the implementation ought to work in time that is poly-logarithmic in the size of the graph, and thus cannot decide "global" properties of the graph. That is, we deal with graphs having $N = 2^n$ vertices, and our procedures run in $\text{poly}(n)$-time.

As in Section 3, we present our results in two categories referring to whether they yield truthful or only almost-truthful implementations. In the case of truthful implementations, we show close-implementations by (polynomial-time) *oracle machines* (which use a random oracle), while bearing in mind that corresponding pseudo-implementations by *ordinary* (probabilistic polynomial-time) machines can be derived using Theorem 2.9. In contrast, in the case of almost-truthful implementations, we work directly with *ordinary* (probabilistic polynomial-time) machines.

6.1 Truthful implementations

The main result of this section is Theorem 6.2, which provide a wide family of monotone graph properties such that there exists a truthful close-implementation of a uniformly selected graph having the corresponding property. Before stating and proving this general result, we consider some concrete special cases.

We first recall that a random graph (i.e., a uniformly distributed N-vertex graph) can be perfectly implemented via an oracle machine that, on input $(u,v) \in [N]\times[N]$ and access to the oracle $f : [N]\times[N] \to \{0,1\}$, returns 0 if $u = v$, $f(u,v)$ if $u < v$, and $f(v,u)$ otherwise. (Indeed, we merely derive a symmetric and non-reflexive version of f.)

Turning to a less trivial example, let us closely-implement a random Bipartite Graph with N vertices on each side. This can be done by viewing the random oracle as two functions, f_1 and f_2, and answering queries as follows:

[22] Indeed, for each function f_s taken from some pseudorandom ensemble $\{f_s : [2^n]\times[2^n] \to \{0,1\}\}_s$, it *may* hold that $f_s(v_s,u) = f_s(u,v_s) = 0$ for all $u \in [2^n]$, where v_s depends arbitrarily on f_s. For example, given a pseudorandom ensemble $\{f_s\}$ consider the ensemble $\{f_{s,v}\}$ such that $f_{s,v}(v,u) = f_{s,v}(u,v) = 0^n$ for all u's, and $f_{s,v}(x,y) = f_s(x,y)$ for all other (x,y)'s.

- The function f_1 is used to closely-implement a random partition of $[2N]$ into two sets of equal size. Specifically, we use f_1 to closely-implement a permutation π over $[2N]$, and let the first part be $S \stackrel{\rm def}{=} \{v : \pi(v) \in [N]\}$. Let $\chi_S(v) \stackrel{\rm def}{=} 1$ if $v \in S$ and $\chi_S(v) \stackrel{\rm def}{=} 0$ otherwise.
- The query (u, v) is answered by 0 if $\chi_S(u) = \chi_S(v)$. Otherwise, the answer equals $f_2(u, v)$ if $u < v$ and $f_2(v, u)$ otherwise.

The above implementation can be adapted to closely-implement a random Bipartite Graph (see details in Appendix B). Viewed in different terms, we have just discussed the implementation of random graphs satisfying certain properties (e.g., being bipartite).

We now turn to Example 3.5 (which specifies a uniformly distributed *connected* graph). In continuation to the discussion in Section 3, we now present a close-implementation that is *truthful*:

Construction 6.1 (implementing a random connected graph): *Use the oracle to closely-implement a random graph, represented by the symmetric and non-reflexive random function $g : [N]\times[N] \to \{0, 1\}$, as well as a random permutation π over $[N]$, which in turn is used to define a Hamiltonian path $\pi(1) \to \pi(2) \to \cdots \to \pi(N)$. Along with π, implement the inverse permutation π^{-1}, where this is done by using Theorem 2.13.*[23] *Answer the query (u, v) by 1 if and only if either $g(u, v) = 1$ or (u, v) is on the Hamiltonian path (i.e., $|\pi^{-1}(u) - \pi^{-1}(v)| = 1$).*

Clearly, the above implementation is truthful with respect to a specification in Example 3.5 (which mandates a connected graph).[24] The implementation is statistically-indistinguishable from the specification, because when making only poly$(\log N)$ queries it is unlikely to hit an edge of the added "Hamiltonian path" (defined by π). (A proof of the latter statement appears below.) A similar strategy can be used for any *monotone* graph property that satisfies the following condition:[25]

(C) The property is satisfied by a family of strongly-constructible sparse graphs. That is, for some negligible function μ (and every N), there exists a perfect implementation of a (single) N-vertex graph with $\mu(\log N) \cdot N^2$ edges that satisfies the property.

We have:

Theorem 6.2 (Construction 6.1, generalized): *Let Π be a* monotone *graph property that satisfies Condition C. Then, there exists a truthful close-implementation (by an oracle machine) of a uniformly distributed graph that satisfies property Π.*

We comment that Condition C implies that a random N-vertex graph is statistically-indistinguishable from a random N-vertex graph having property Π. This fact, which may be of independent interest, is stated and proved first.

Lemma 6.3 *Let Π be a* monotone *graph property that is satisfied by some N-vertex graph having $\epsilon \cdot \binom{N}{2}$ edges. Then, any machine that makes at most q adjacency queries to a graph, cannot distinguish a random N-vertex graph from a uniformly distributed N-vertex graph that satisfies Π, except than with probability $O(q \cdot \sqrt{\epsilon}) + q \cdot N^{-(1-o(1))}$.*

Proof: As in [23, Sec. 4], without loss of generality, we may confine ourselves to analyzing machines that inspect a random induced subgraph. That is, since both graph classes are closed under isomorphism, it suffices to consider the statistical difference between the following two distributions:

1. The subgraph of a uniformly distributed N-vertex graph induced by a uniformly selected set of $s \stackrel{\rm def}{=} q + 1$ vertices.

[23] That is, we use a truthful close-implementation of Example 2.4. In fact, we only need π^{-1}, and so the truthful close-implementation of Example 2.3 (as stated in Theorem 2.12) actually suffices.

[24] Indeed, Construction 6.1 actually implements a (random) Hamiltonian graph (by virtue of the "forced Hamiltonian path" defined by π).

[25] Here C stands for "condition".

2. The same vertex-induced subgraph (i.e., induced by a random set of s vertices) of a uniformly distributed N-vertex graph that satisfies property Π.

Clearly, distribution (1) is uniform over the set of s-vertex graphs, and so we have to show that approximately the same holds for Distribution (2). Let $T \stackrel{\rm def}{=} \binom{N}{2}$ and $M \stackrel{\rm def}{=} \epsilon T$, and let G_0 be an N-vertex graph with M edges that satisfies property Π. Consider the set of all graphs that can be obtained from G_0 by adding $\frac{T-M}{2}$ edges. The number of these graphs is

$$\binom{T-M}{\frac{T-M}{2}} = \frac{2^{T-M}}{\Theta(\sqrt{T-M})} > 2^{T-M-O(1)-\frac{1}{2}\cdot\log_2 T}$$

That is, this set contains at least a $2^{-(M+O(1)+(\log_2 T)/2)} = 2^{-\epsilon'\cdot T}$ fraction of all possible graphs, where $\epsilon' \stackrel{\rm def}{=} \epsilon + ((\log_2 T)/2T)$. Let $X = X_1 \cdots X_T \in \{0,1\}^T$ be a random variable that is uniformly distributed over the set of all graphs that satisfy property Π. Then X has entropy at least $T - \epsilon' T$ (i.e., $\mathbf{H}(X) \geq T - \epsilon' T$). It follows that $\frac{1}{T}\sum_{i=1}^{T} \mathbf{H}(X_i|X_{i-1}\cdots X_1) \geq 1-\epsilon'$, where the index i ranges over all unordered pairs of elements of $[N]$. (Indeed, we assume some fixed order on these pairs.) Letting $e_j(S)$ denote the $j^{\rm th}$ pair in the set $\{(u,v)\in S\times S : u < v\}$, we are interested in the expected value of $\sum_{j=1}^{\binom{s}{2}} \mathbf{H}(X_{e_j(S)}|X_{e_{j-1}(S)}\cdots X_{e_1(S)})$, where S is a uniformly selected set of t vertices. Clearly,

$$\mathbf{H}(X_{e_j(S)}|X_{e_{j-1}(S)}\cdots X_{e_1(S)}) \geq \mathbf{H}(X_{e_j(S)}|X_{e_j(S)-1}\cdots X_1)$$

and so

$$\mathbf{E}_S\left[\sum_{j=1}^{\binom{s}{2}} \mathbf{H}(X_{e_j(S)}|X_{e_{j-1}(S)}\cdots X_{e_1(S)})\right] \geq \binom{s}{2}\cdot(1-\epsilon')$$

because for a uniformly distributed $j \in [\binom{s}{2}]$ it holds that $\mathbf{E}_{S,j}\left[\mathbf{H}(X_{e_j(S)}|X_{e_j(S)-1}\cdots X_1)\right]$ equals $\mathbf{E}_i\left[\mathbf{H}(X_i|X_{i-1}\cdots X_1)\right]$, where i is uniformly distributed in $[T]$. Thus, for a random s-subset S, letting $Y_S = (X_{(u,v)})_{\{(u,v)\in S\times S: u<v\}}$, we have $\mathbf{E}_S[Y_S] \geq t - \epsilon''$, where $t \stackrel{\rm def}{=} \binom{s}{2}$ and $\epsilon'' \stackrel{\rm def}{=} t\epsilon'$. It follows (see Appendix C) that the statistical difference of Y_S from the uniform distribution over $\{0,1\}^t$ is at most $O(\sqrt{\epsilon''})$, which in turn equals $O(q\sqrt{\epsilon + T^{-(1-o(1))}})$. The lemma follows. ■

Proof of Theorem 6.2: Let $H = ([N], E)$ be a graph satisfying Condition C. In particular, given $(u,v) \in [N]\times[N]$, we can decide whether or not $(u,v) \in E$ in polynomial-time. Then, using the graph H instead of the Hamiltonian path in Construction 6.1, we implement a (random) graph satisfying property Π. That is, we answer the query (u,v) by 1 if and only if either $g(u,v) = 1$ or (u,v) is an edge in (the "forced" copy of) H (i.e., $(\pi^{-1}(u), \pi^{-1}(v)) \in E$). Since Π is a monotone graph property, the instances of the implementation always satisfy the property Π, and thus the implementation is truthful. Furthermore, by Condition C and the fact that π is a close-implementation of a random permutation, the probability that a machine that queries the implementation for poly($\log N$) times hits an edge of H is negligible in $\log N$. Thus, such a machine cannot distinguish the implementation from a random graph. Using Lemma 6.3 (with $\epsilon = \mu(\log N)$ and $q = \mathrm{poly}(\log N)$), the theorem follows. ■

Examples: Indeed, monotone graph properties satisfying Condition C include Connectivity, Hamiltonicity, k-Connectivity (for every fixed k)[26], containing any fixed-size graph (e.g., containing a triangle or a 4-clique or a $K_{3,3}$ or a 5-cycle), having a perfect matching, having diameter at most 2, containing a clique of size at least $\log_2 N$, etc. All the foregoing properties are satisfied, with overwhelmingly high probability, by a random graph. However, *Theorem 6.2 can be applied also to (monotone) properties that are not satisfied by a random graph*; a notable example is the property of containing a clique of size at least $\sqrt{N}$.

[26] In fact, we may have $k = k(N) = \mu(\log N)\cdot N$ for any negligible function μ. The sparse graph may consist of a complete bipartite graph with $k(N)$ vertices on one side and $N - k(N) \approx N$ vertices on the other side.

6.2 Almost-truthful implementations

Here we consider almost-truthful implementations of specifications that refer to uniformly selected graphs that have various properties that are satisfied by random graphs. (Needless to say, we will focus on properties that are not covered by Theorem 6.2.)

We start by noting that if we are willing to settle for almost-truthful implementations by oracle machines then all properties that hold (with sufficiently high probability) for random graphs can be handled easily. Specifically:

Proposition 6.4 *Let Π be any graph property that is satisfied by all but a negligible (in $\log N$) fraction of the N-vertex graphs. Then, there exists an almost-truthful close-implementation (by an oracle machine) of a uniformly distributed graph that satisfies property Π.*

Indeed, the implementation is by a random graph (which in turn is implemented via a random oracle). Note, however, that it is not clear what happens if we replace the random graph by a pseudorandom one (cf. Theorem 2.11). Furthermore, the proof of Theorem 2.11 can be extended to show that there exist graph properties that are satisfied by random graphs but do not have an almost-truthful implementation *by an ordinary machine*.[27] In light of the above, we now focus on almost-truthful implementations *by ordinary machines*. As we shall see, that the technique underlying Construction 6.1 can be used also when the following relaxed form of Condition (C) holds:

(C') For some negligible function μ (and every N), there exists an *almost-truthful* implementation (by ordinary machines) of a distribution over N-vertex graphs that satisfy the property and have at most $\mu(\log N) \cdot N^2$ edges.

Indeed, we may obtain a variant of Theorem 6.2 stating that, assuming the existence of one-way functions, *for every monotone graph property that satisfies Condition C', there exists an almost-truthful pseudo-implementation (by an ordinary machine) of a uniformly distributed graph that satisfies property* Π. However, our main focus in the current subsection will be on non-monotone graph properties (e.g., having a max-clique of a certain size), and in this case we cannot apply Lemma 6.3. Instead, we shall use the following observation, which refer to properties that are satisfied by random graphs (e.g., having a max-clique of logarithmic size).

Proposition 6.5 *Let Π be any graph property that is satisfied by all but a negligible (in $\log N$) fraction of the N-vertex graphs. Let S be the specification that uniformly selects an N-vertex graph that satisfies property Π and answers edge-queries accordingly, and let I be any pseudo-implementation of a uniformly distributed N-vertex graph. Then I is a pseudo-implementation of S.*

Indeed, Proposition 6.5 holds because the first hypothesis implies that S is computationally indistinguishable from a truly random graph, whereas the second hypothesis asserts that I is computationally indistinguishable from a truly random graph.

Max-clique and chromatic number. We consider the construction of pseudorandom graphs that preserve the max-clique and chromatic number of random graphs.

Theorem 6.6 *Let $c(N) = (2 - o(1)) \log_2 N$ be the largest integer i such that the expected number of cliques of size i in a random N-vertex graph is larger than one. Assuming the existence of one-way functions, there exist almost-truthful pseudo-implementations, by ordinary machines, of the following specifications:*

[27] The proof of Theorem 2.11 relates to the Kolmogorov Complexity of the function (or graph). In order to obtain a graph property, we consider the minimum value of the Kolmogorov Complexity of any isomorphic copy of the said graph, and consider the set of graphs for which this quantity is greater than $N^2/4$. The latter property is satisfied by all but at most $2^{N^2/4} \cdot (N!) \ll 2^{N^2/3}$ graphs. On the other hand, the property cannot be satisfied by an instance of an implementation via an ordinary machine. Thus, any implementation (regardless of "quality") must be non-truthful (to the specification) in a strong sense.

1. A random graph of Max-Clique $c(N) \pm 1$: *The specification uniformly selects an N-vertex graph having maximum clique size $c(N) \pm 1$, and answers edge-queries accordingly.*

2. A random graph of Chromatic Number $(1 \pm o(1)) \cdot N/c(N)$: *The specification uniformly selects an N-vertex graph having Chromatic Number $(1 \pm \log_2^{-1/3} N) \cdot N/c(N)$, and answers edge-queries accordingly.*

That is, we are required to implement random-looking graphs having certain properties. Indeed, a random N-vertex graph has the above two properties with probability at least $1 - N^{-0.99}$ (cf. [8]). Thus, a random graph provides an almost-truthful close-implementation (by an oracle machine) of a uniformly selected graph having each of these properties, but it is not clear what happens when we replace the random oracle by a pseudorandom function. (In fact, one can easily construct pseudorandom functions for which the replacement yields a graph with a huge clique or alternatively, with a very small chromatic number.) Note that Theorem 6.6 does not follow from Theorem 6.2, because the properties at hand are not monotone.[28] Thus, a different approach is needed.

Proof: We start with Part 1. We define the adjacency function $g^{\rm clique} : [N] \times [N] \to \{0,1\}$ of a graph by XORing a pseudorandom function f with a k-wise independent function f' (i.e., $g^{\rm clique}(v,w) = f(v,w) \oplus f'(v,w)$), where $k \stackrel{\rm def}{=} 4n^2$ (and $n = \log_2 N$).[29] Recall that such k-wise independent functions can be constructed based on kn random bits. The resulting function $g^{\rm clique}$ is both k-wise independent and computationally indistinguishable from a random graph (analogously to the construction in [26]). In particular, using the pseudorandomness of $g^{\rm clique}$ and the fact that a random graph violates the specification with negligible probability (in $\log N$), it follows that $g^{\rm clique}$ pseudo-implements a uniformly distributed N-vertex graph having max-clique $c(N) \pm 1$. (Indeed, the foregoing argument relies on Proposition 6.5.)

Next, we use the k-wise independence of $g^{\rm clique}$ in order to show that $g^{\rm clique}$ is almost-truthful. The key observation is that the Bollobás–Erdös analysis [9] of the size of the max-clique in a random graph only refers to the expected number of cliques of size $c(N) \pm 2$ and to the variance of this random variable. Thus, this analysis only depends on the randomness of edges within pairs of $(c(N)+2)$-subsets of vertices; that is, a total of $2 \cdot \binom{c(N)+2}{2} < (c(N)+2)^2 = (4 - o(1)) \cdot n^2$ vertex-pairs. Hence, the analysis continues to hold for $g^{\rm clique}$ (which is $4n^2$-independent), and so with overwhelming probability $g^{\rm clique}$ has max-clique size $c(N) \pm 1$. It follows that $g^{\rm clique}$ provides an almost-truthful pseudo-implementation of a random N-vertex graph with max-clique size $c(N) \pm 1$.

We now turn to Part 2. We define the adjacency function $g^{\rm color} : [N] \times [N] \to \{0,1\}$ of a graph by taking the bit-wise conjunction of the graph $g^{\rm clique}$ with a function h selected uniformly in a set H (defined below); that is, $g^{\rm color}(v,w) = 1$ iff $g^{\rm clique}(v,w) = h(v,w) = 1$. Intuitively, each function $h \in H$ forces a cover of $[N]$ by $N/c(N)$ independent sets, each of size $c(N)$, and so the chromatic number of $g^{\rm color}$ is at most $N/c(N)$. On the other hand, by symmetry (of edges and non-edges), the graph $g^{\rm clique}$ doesn't only exhibit clique-number $c(N) \pm 1$ (which is irrelevant in this part) but also has independence-number $c(N) \pm 1$ (with overwhelming probability). We will use the latter fact to show that, since each $h \in H$ only has independent sets of size $c(N)$, taking the conjunction with $g^{\rm clique}$ is unlikely to create an independent set of size $c(N)+2$. Thus, the chromatic number of $g^{\rm color}$ is at least $N/(c(N)+1)$. Details follow.

Each function $h \in H$ partitions $[N]$ into $\chi(N) \stackrel{\rm def}{=} \lceil N/c(N) \rceil$ *forced independent sets*, where each set (except the last) is of size $c(N)$. We define $h(v,w) = 1$ if and only if v and w belong to different sets; Thus, such h causes each of these vertex-sets to be an independent set in $g^{\rm color}$. The functions in H differ only in the partitions that they use. It turns out that it suffices to use "sufficiently random" partitions. Specifically, we use $H = \{h_r\}_{r \in R}$, where $R = \{r \in [N] : \gcd(r,N) = 1\}$, and consider for each shift $r \in R$ the partition into forced independent sets $(S_r^{(1)}, ..., S_r^{(\chi(N))})$, where $S_r^{(i)} = \{(i \cdot c(N) + j) \cdot r \bmod N : j = 1, ..., c(N)\}$ for $i < \chi(N)$ (and $S_r^{(\chi(N))}$ contains the $N - (\chi(N)-1) \cdot c(N)$ remaining vertices). Note that the condition $\gcd(r,N) = 1$ ensures that this is indeed a proper partition of the vertex-set $[N]$. Thus, $h_r(v,w) = 1$ if

[28] For the coloring property, Condition C does not hold either.

[29] As in other places, we actually mean symmetric and non-reflexive functions that are obtained from the values of the basic functions at values (u,v) such that $u < v$.

and only if v and w do not reside in the same forced independent set $S_r^{(i)}$ (i.e., $h_r(v,w) = 0$ implies that $|v-w| \equiv jr \pmod N$ for some $j \in \{1,...,(c(N)-1)\}$).

To establish the pseudorandomness of the implementation, we first note that $g^{\rm color}$ is computationally indistinguishable from $g^{\rm clique}$ (and consequently $g^{\rm color}$ retains $g^{\rm clique}$'s indistinguishability from a random graph). Indeed, it can be shown that no efficient observer is likely to make a query (v,w) that is affected by h_r, because $h_r(v,w) = 0$ yields at most $2(c(N)-1) = \Theta(\log N)$ candidates for r, which in turn is selected uniformly in the set R, where $|R| = N^{\Omega(1)}$. In addition, a random graph has only a negligible probability (in $\log N$) of having chromatic number different from $(1 \pm \log_2^{-1/3} N) \cdot N/c(N)$. Combining all this with Proposition 6.5 implies the pseudorandomness of the implementation (w.r.t the specification).

We now turn to the almost-truthfulness requirement. First note that the chromatic number of $g^{\rm color}$ is at most $\chi(N)$, because its vertex-set is covered by $\chi(N)$ independent sets. On the other hand, we will show that with overwhelming probability, the graph $g^{\rm color}$ does not contain an independent set of size $c(N)+2$. Thus, the chromatic number of $g^{\rm color}$ is at least $N/(c(N)+1) > (1-(2/c(N))\cdot\chi(N)$, and so $g^{\rm color}$ is an almost-truthful pseudo-implementation of the desired specification, and the entire theorem follows. Thus, it is left to show that the independence-number of $g^{\rm color}$ is at most $c(N)+1$. The argument proceeds as follows. We fix any $h = h_r \in H$ (so the forced independent sets $S_r^{(j)}$ are fixed) and show that deleting edges as instructed by a k-wise independent function (i.e., by $g^{\rm clique}$) is unlikely to induce a $c(N)+2$ independent set. Note that the various candidate independent sets differ with respect to their intersection with the forced independent sets $S_r^{(j)}$, and the analysis has to take this into account. For example, if the candidate independent set does not contain two vertices of the same set $S_r^{(j)}$, which is indeed the typical case, then the analysis of $g^{\rm clique}$ suffices. At the other extreme, there is the case that the candidate independent set contains all vertices of some set $S_r^{(j)}$. In this case, we only have either $2c(N)$ or $2c(N)+1$ random events (i.e., regarding edges between $S_r^{(j)}$ and the other two vertices), but the number of possibilities that correspond to this case is smaller than N^3, and so the total probability for the corresponding bad event is less than $N^3 \cdot 2^{-2c(N)} = N^{-1+o(1)}$. The full analysis, given in Appendix C, consists of a rather straightforward and tedious case analysis. ■

Combining properties of random graphs. So far, we considered several prominent properties that are satisfied (w.h.p.) by random graphs, and provided pseudo-implementations of uniformly distributed graphs that satisfy each of these properties separately. Next, we discuss a construction of pseudorandom graphs that simultaneously satisfy all those properties of random graphs.

Theorem 6.7 *Let $c(N) = (2-o(1))\log_2 N$ be as in Theorem 6.6. Assuming the existence of one-way functions, there exists an almost-truthful pseudo-implementation, by an ordinary machine, of the specification that uniformly selects an N-vertex graph that satisfies the following four properties:*

1. *Being Hamiltonian.*
2. *Having Clique Number $c(N) \pm 1$.*
3. *Having Independence Number $c(N) \pm 1$.*
4. *Having Chromatic Number $(1 \pm \log_2^{-1/3} N) \cdot N/c(N)$.*

The specification answers edge-queries accordingly.

Recall that being Hamiltonian implies being connected as well has containing a Perfect Matching.

Proof: Consider the following implementation that merely adds a (carefully chosen) random looking Hamiltonian cycle $g^{\rm Ham}$ to the pseudorandom graph $g^{\rm color}$ that was defined in the proof of Theorem 6.6. That is, we define our adjacency function $g^{\rm combine} : [N] \times [N] \to \{0,1\}$ of a graph as the bit-wise disjunction of $g^{\rm color}$ with the adjacency function $g^{\rm Ham}$ (specified below); i.e., $g^{\rm combine}(v,w) = 1$ if and only if either $g^{\rm color}(v,w) = 1$ or $g^{\rm Ham}(v,w) = 1$. Towards defining $g^{\rm Ham}$, recall that in $g^{\rm color}$ the vertices are covered with $\chi(N) \stackrel{\rm def}{=} \lceil N/c(N) \rceil$ disjoint independent sets $\{S_r^{(i)}\}_{i=1}^{\chi(N)}$, where each set (except the last) is of size $c(N)$ and where the sets are defined using a random shift r uniformly chosen in $R = \{r' \in [N] : \gcd(r', N) = 1\}$. We now define $g^{\rm Ham}$ such that $g^{\rm Ham}$ does not violate any of the forced independent sets of $g^{\rm color}$, and consequently the $\chi(N)$ upper-bound on the chromatic number of $g^{\rm color}$ is retained by $g^{\rm combine}$. Specifically, we define $g^{\rm Ham}$ using *the same* random shift r that is used to define the forced independent sets $S_r^{(i)}$: using an arbitrary integer $d \in [c(N), N - c(N)]$ that satisfies $\gcd(d, N) = 1$, we set $g_r^{\rm Ham}(v, w) = 1$ if and only if $w = (v \pm dr) \bmod N$.

We first establish the pseudorandomness of the implementation. We note that $g^{\rm combine}$ is computationally indistinguishable from $g^{\rm color}$, because no efficient observer is likely to make a query (v, w) that is affected by $g_r^{\rm Ham}$. Indeed, r is selected uniformly in the set R of size $|R| = N^{\Omega(1)}$, while $g_r^{\rm Ham}(v, w) = 1$ implies only two candidates for r (a single candidate for each of the possible cases of either $(w = v + dr) \bmod N$ or $(w = v - dr) \bmod N$). Consequently, the computational indistinguishability of $g^{\rm color}$ from a random graph (which was established during the proof of Theorem 6.6) is preserved by $g^{\rm combine}$. We next recall (cf. [8]) that, only with negligible probability (in $\log N$), a random graph fails to exhibit properties 1–4 listed above. Hence, the pseudorandomness of the implementation (w.r.t the specification) follows from Proposition 6.5.

We now turn to establish the almost-truthfulness claim. Regarding Hamiltonicity, note that our selection of r and d (which satisfies $\gcd(r, N) = 1 = \gcd(d, N)$) guarantees that the graph $g_r^{\rm Ham}$ is indeed an Hamiltonian cycle (because $dr, 2dr, 3dr, ..., Ndr$ are all distinct modulo N). It follows that $g^{\rm combine}$ is always Hamiltonian.

We now handle the independence number and chromatic number. Clearly, since $g^{\rm combine}$ is obtained by adding edges to $g^{\rm color}$, the former retains $g^{\rm color}$'s properties of almost surely having independence number at most $c(N) + 1$ and chromatic number at least $N/(c(N) + 1)$. In addition, by the definition of the forced independent sets $S_r^{(i)}$, an arbitrary pair of vertices (v, w) belongs to the same $S_r^{(i)}$ only if $w = (v \pm jr) \bmod N$ where $j \in \{1, ..., c(N) - 1\}$. On the other hand, $g_r^{\rm Ham}(v, w) = 1$ implies that $w = (v + dr) \bmod N$ or $w = (v - dr) \bmod N$ where $c(N) \leq d \leq N - c(N)$. Since $\gcd(r, N) = 1$ the above implies that the edges of the Hamiltonian cycle $g^{\rm Ham}$ never violate any of the forced independent sets of $g^{\rm color}$. Thus, as the forced independent sets are of size $c(N)$, and since these sets force a cover of $[N]$ with $\lceil N/c(N) \rceil$ independent sets, it follows that $g^{\rm combine}$ achieves independence number at least $c(N)$ and chromatic number at most $\lceil N/c(N) \rceil$ (just as $g^{\rm color}$ does).

The last property to consider is the clique number; that is, we now show that $g^{\rm combine}$ has clique number $c(N) \pm 1$ (almost surely). The argument is based on the fact (taken from the proof of Theorem 6.6) that $g^{\rm clique}$ has clique number $c(N) \pm 1$ almost surely. Indeed, let $c = c(N)$. As $g^{\rm color}$ is obtained by omitting edges from $g^{\rm clique}$ and $g^{\rm combine}$ is (later) obtained by adding edges to $g^{\rm color}$, it suffices to establish a $c - 1$ lower bound on the clique number of $g^{\rm color}$ and a $c + 1$ upper bound on the clique number of $g^{\rm combine}$. To this end we fix (again) the random shift r (which specifies both the forced independent sets of $g^{\rm color}$ as well as the Hamiltonian cycle $g^{\rm Ham}$), and establish the desired bounds when the probabilities are taken only over the k-wise independent graph $g^{\rm clique}$.

Towards proving the lower bound (on the clique number of $g^{\rm color}$), let $X^{\rm clique}$ and $X^{\rm color}$ denote the random variables that count the number of $(c-1)$-cliques in $g^{\rm clique}$ and in $g^{\rm color}$, respectively. By Chebyshev's inequality the probability of having no $(c - 1)$-cliques in $g^{\rm color}$ is upper bounded by $\frac{\mathsf{var}(X^{\rm color})}{(E(X^{\rm color}))^2}$. Since it is known (see [9]) that $\frac{\mathsf{var}(X^{\rm clique})}{(E(X^{\rm clique}))^2}$ is negligibly small (in $\log N$), it suffices to show that

$$\frac{\mathsf{var}(X^{\rm color})}{(E(X^{\rm color}))^2} = O\left(\frac{\mathsf{var}(X^{\rm clique})}{(E(X^{\rm clique}))^2}\right). \tag{2}$$

We first argue that $\mathsf{var}(X^{\rm color}) \leq \mathsf{var}(X^{\rm clique})$. Let $\mathbf{T}$ denote the collection of all subsets of vertices of cardinality $c - 1$, and let $\mathbf{T}^{\rm color} \subset \mathbf{T}$ denote only those subsets that contain at most one vertex from each forced independent set; that is, $\mathbf{T}$ contains exactly all "potential cliques" of $g^{\rm clique}$, while $\mathbf{T}^{\rm color}$ contains only the "potential cliques" of $g^{\rm color}$). For each $T \in \mathbf{T}$, let $X_T^{\rm clique}$ and $X_T^{\rm color}$ denote the random variables that indicate whether T induces a clique in $g^{\rm clique}$ and in $g^{\rm color}$, respectively. Since, for any $T, T' \in \mathbf{T}^{\rm color}$, it holds that T induces a clique in $g^{\rm clique}$ if and only if it induces a clique in $g^{\rm color}$, we get $\mathsf{var}(X_T^{\rm clique}) = \mathsf{var}(X_T^{\rm color})$

and $\mathsf{cov}(X_T^{\rm clique}, X_{T'}^{\rm clique}) = \mathsf{cov}(X_T^{\rm color}, X_{T'}^{\rm color})$. Since all the terms in the expansion

$$\mathsf{var}(X^{\rm color}) = \sum_{T\in\mathbf{T}^{\rm color}} \mathsf{var}(X_T^{\rm color}) + \sum_{T\neq T'\in\mathbf{T}^{\rm color}} \mathsf{cov}(X_T^{\rm color}, X_{T'}^{\rm color}),$$

also appear in the expansion of $\mathsf{var}(X^{\rm clique})$, and as all terms in in the expansion of $\mathsf{var}(X^{\rm clique})$ are non-negative, we get $\mathsf{var}(X^{\rm color}) \leq \mathsf{var}(X^{\rm clique})$.

Next we show that $E(X^{\rm color}) = (1-o(1)) \cdot E(X^{\rm clique})$. First note that $E(X^{\rm clique}) = \binom{N}{c-1} \cdot 2^{-\binom{c-1}{2}}$. On the other hand, the number of potential $(c-1)$-cliques in $g^{\rm color}$ is lower-bounded by $L \stackrel{\rm def}{=} \binom{\lfloor \frac{N}{c} \rfloor}{c-1} \cdot c^{c-1}$, because there are $\lfloor \frac{N}{c} \rfloor$ forced independent sets $S^{(i)}$ of size c, and a potential clique can be specified by first choosing $c-1$ of these sets $S^{(i)}$, and then choosing a single vertex from each set. Next note that all relevant edges are determined only by the $4n^2$-wise independent graph $g^{\rm clique}$, and so $E(X^{\rm color}) \geq L \cdot 2^{-\binom{c-1}{2}}$. Since $L = \binom{\lfloor \frac{N}{c} \rfloor}{c-1} c^{c-1} = (1-o(1)) \cdot \binom{N}{c-1}$, we get $E(X^{\rm color}) \geq (1-o(1)) \cdot \binom{N}{c-1} \cdot 2^{-\binom{c-1}{2}}$, which in turn equals $(1-o(1)) \cdot E(X^{\rm clique})$. Having established Eq. (2), we conclude that (with very high probability) the $c-1$ lower bound on the clique number of $g^{\rm color}$ holds.

Our final task is to establish a $c+1$ upper bound on the clique number of $g^{\rm combine}$; that is, to show that for $c' \stackrel{\rm def}{=} c(N)+2$, with high probability $g^{\rm combine}$ contains no c'-cliques. Let's first consider $g^{\rm color}$. Recall that by [9], $g^{\rm clique}$ has a negligible probability (in $\log N$) of having a c'-clique. As $g^{\rm color}$ is obtained by omitting edges from $g^{\rm clique}$ the same holds for $g^{\rm color}$ as well. Consequently, as $g^{\rm combine}$ is obtained by adding a single Hamiltonian cycle $g^{\rm Ham}$ to $g^{\rm color}$, it suffices to give a negligible upper-bound only on the probability that $g^{\rm combine}$ contains a c'-clique that intersects $g^{\rm Ham}$ (in at least one edge). This is done by showing that the expected number of the latter cliques is negligible (in $\log N$).[30]

We use the following terminology. Given a vertex-set V of size c' (i.e., a potential clique), we say that a vertex $w \in V$ is a follower-vertex if its predecessor in $g^{\rm Ham}$ is in V (i.e., if $w - dr \bmod N$ is in $\{v \bmod N : v \in V\}$). Let $\mathbf{V}_k$ denote the collection of all vertex-sets V of size c' that have exactly k follower-vertices. We now bound $\mathbf{E}_k$, the expected number of cliques induced by vertex-sets $V \in \mathbf{V}_k$. For $V \in \mathbf{V}_k$, the number of edges of $g^{\rm Ham}$ that have both endpoints in V is k. Since the rest of the edges of V are decided by the $4n^2$-wise independent graph $g^{\rm clique}$, the probability that V induces a $(c-1)$-clique in $g^{\rm combine}$ is at most $2^{-\binom{c'}{2}+k}$. Next observe that $|\mathbf{V}_k| \leq \binom{N}{c'-k} \cdot (c'-1)^k$, because a set $V \in \mathbf{V}_k$ is defined by the choice of $c'-k$ non-follower and k (successive) choices of followers (where the $i^{\rm th}$ follower is selected as following one of the $c'-k+(i-1) \leq c'-1$ vertices selected so far). Thus

$$\mathbf{E}_k \leq \binom{N}{c'-k} \cdot (c'-1)^k \cdot 2^{-\binom{c'}{2}+k} = \binom{N}{c'} \cdot (N^{-1+o(1)})^k \cdot 2^{-\binom{c'}{2}} \ll \binom{N}{c'} \cdot 2^{-\binom{c'}{2}},$$

where the latter expression is upper-bound by $N^{-\Omega(1)}$ (see [9], while recalling that $c' = c(N)+2$). If follows that $\sum_{k=1}^{c'-1} \mathbf{E}_k$ is negligible (in $\log N$). This establishes the upper-bound on the clique-number of $g^{\rm combine}$, which completes the proof of the entire theorem. ■

High connectivity. One property of random graphs that was left out of Theorem 6.7 is having high (global) connectivity property: Indeed, in a random N-vertex graph, every pair of vertices is connected by at least $(1-o(1))N/2$ vertex-disjoint paths. One interesting question is to provide an almost-truthful pseudo-implementation of a uniformly distributed graph having this high (global) connectivity property. Unfortunately, at the time this research was conducted, we did not know how to do this.[31] A second best thing may be to provide an almost-truthful pseudo-implementation of a random graph for which almost all pairs of vertices enjoy this "high connectivity" property.

[30]Recall that we fixed the random shift r (which specifies both the forced independent sets of $g^{\rm color}$ as well as the enforced Hamiltonian path $g_r^{\rm Ham}$), and so probabilities are taken only over the k-wise independent choices of the edges of $g^{\rm clique}$.

[31]We mention that subsequent work by Alon and Nussboim [3] has provided the implementation we failed to achieve (i.e., an almost-truthful pseudo-implementation of a uniformly distributed graph having a high global connectivity property).

Theorem 6.8 *For every positive polynomial p, assuming the existence of one-way functions, there exists an almost-truthful pseudo-implementation by an ordinary machine of the following specification. The specifying machine selects a graph that is uniformly distributed among all N-vertex graphs for which all but at most an* $\epsilon(N) \stackrel{\text{def}}{=} 1/p(\log_2 N)$ *fraction of the vertex pairs are connected by at least* $(1-\epsilon(N)) \cdot N/2$ *vertex-disjoint paths. Edge-queries are answered accordingly.*

Interestingly, the same implementation works for all polynomials p; that is, the implementation is independent of p, which is only needed for the definition of the specification. In fact, in contrast to all other implementations presented in this work, the implementation used in the proof of Theorem 6.8 is the straightforward one: It uses a pseudorandom function to define a graph in the obvious manner. The crux of the proof is in showing that this implementation is computationally-indistinguishable from the foregoing specification.

Proof: We use a pseudorandom function to define a graph $G = ([N], E)$ in the straightforward manner, and answer adjacency queries accordingly. This yields a pseudo-implementation of a truly random graph, which in turn has the strong connectivity property (with overwhelmingly high probability). Fixing a polynomial p and $\epsilon \stackrel{\text{def}}{=} \epsilon(N) \stackrel{\text{def}}{=} 1/p(\log_2 N)$, we prove that this implementation is almost-truthful to the corresponding specification. That is, we show that, with overwhelmingly high probability, all but at most an ϵ fraction of the vertex pairs are connected via $(1-\epsilon) \cdot N/2$ vertex-disjoint paths. We will show that if this is not the case, then we can distinguish a random graph (or function) from a pseudorandom one.

Suppose towards the contradiction that, with non-negligible probability, a pseudorandom graph violates the desired property. Fixing such a graph, $G = ([N], E)$, our hypothesis means that at least an ϵ fraction of the vertex-pairs are connected (in G) by fewer than $(1-\epsilon) \cdot N/2$ vertex-disjoint paths. Consider such a generic pair, denoted (u, v), and define $S_0 \stackrel{\text{def}}{=} \Gamma_G(u) \cap \Gamma_G(v)$, $S_1 \stackrel{\text{def}}{=} \Gamma_G(u) \setminus \Gamma_G(v)$, and $S_2 \stackrel{\text{def}}{=} \Gamma_G(v) \setminus \Gamma_G(u)$, where $\Gamma_G(w) \stackrel{\text{def}}{=} \{x \in [N] : (w, x) \in E\}$. Note that if G were a random graph then we would expect to have $|S_0| \approx |S_1| \approx |S_2| \approx N/4$. Furthermore, we would expect to see a large (i.e., size $\approx N/4$) matching in the induced bipartite graph $B = ((S_1, S_2), E \cap (S_1 \times S_2))$; that is, the bipartite graph having S_1 on one side and S_2 on the other. So, the intuitive idea is to test that both these conditions are satisfied in the pseudorandom graph. If they do then u and v are "sufficiently connected". Thus, the hypothesis that an ϵ fraction of the vertex-pairs are no "sufficiently connected" implies a distinguisher (by selecting vertex-pairs at random and testing the above two properties). The problem with the foregoing outline is that it is not clear how to *efficiently* test that the aforementioned bipartite graph B has a sufficiently large matching.

To allow an efficient test (and thus an efficient distinguisher), we consider a more stringent condition (which would still hold in a truly random graph). We consider a fixed partition of $[N]$ into $T \stackrel{\text{def}}{=} N/m$ parts, $(P_1, ..., P_T)$, such that $|P_i| = m = \text{poly}(n/\epsilon)$, where $n = \log_2 N$. (For example, we may use $P_i = \{(i-1)m + j : j = 1, ..., m\}$.) If G were a random graph then, with overwhelmingly high probability (i.e., at least $1 - \exp(-m^{1/O(1)}) > 1 - \exp(-n^2)$), we would have $|S_0 \cap P_i| = (m/4) \pm m^{2/3}$ for all the i's. Similarly for S_1 and S_2. Furthermore, with probability at least $1 - \exp(-n^2)$, each of the bipartite graphs B_i induced by $(P_i \cap S_1, P_i \cap S_2)$ would have a matching of size at least $(m/4) - m^{2/3}$. The key point is that we can afford to test the size of the maximum matching in such a bipartite graph, because it has $2m = \text{poly}(n)$ vertices.

Let us wrap-up things. If a pseudorandom graph does not have the desired property then at least ϵ fraction of its vertex-pairs are connected by less than $(1-\epsilon)N/2$ vertex-disjoint paths. Thus, sampling $O(1/\epsilon)$ vertex-pairs, we hit such a pair with constant probability. For such a vertex-pair, we consider the sets $S_{i,0} \stackrel{\text{def}}{=} P_i \cap S_0$, $S_{i,1} \stackrel{\text{def}}{=} P_i \cap S_1$ and $S_{i,2} \stackrel{\text{def}}{=} P_i \cap S_2$, for $i = 1, ..., T$. It must be the case that either $\epsilon/2$ fraction of the $S_{0,i}$'s are of size less than $(1 - (\epsilon/2)) \cdot (m/4)$ or that $\epsilon/2$ fraction of the bipartite subgraphs (i.e., B_i's) induced by the pairs $(S_{1,i}, S_{2,i})$ have no matching of size $(1-(\epsilon/2)) \cdot (m/4)$, because otherwise this vertex-pair is sufficiently connected merely by virtue of these $S_{0,i}$'s and the large matchings in the B_i's.[32] We use $m > (8/\epsilon)^3$ so to guarantee that $(m/4) - m^{2/3} > (1-(\epsilon/2))(m/4)$, which implies that (for at least an $\epsilon/2$ fraction of the i's) some quantity (i.e., either $|S_{0,i}|$ or the maximum matching in B_i) is strictly larger in a random graph than in a pseudorandom graph. Now, sampling $O(1/\epsilon)$ of the i's, we declare the graph to be

[32] That is, we get at least $((1-(\epsilon/2)) \cdot T) \cdot ((1-(\epsilon/2)) \cdot (m/4)) > (1-\epsilon)(N/4)$ paths going through S_0, and the same for paths that use the maximum matchings in the various B_i's.

random if all the corresponding $S_{0,i}$'s have size at least $(m/4) - m^{2/3}$ and if all the corresponding bipartite graphs B_i's have a maximum matching of size at least $(m/4) - m^{2/3}$. Thus, we distinguish a random function from a pseudorandom function, in contradiction to the definition of the latter. The theorem follows. ■

Maximum Matching in most induced bipartite graphs: The proof of Theorem 6.8 can be adapted to prove the following:

Theorem 6.9 *For every positive polynomial p, assuming the existence of one-way functions, there exists an almost-truthful pseudo-implementation by an ordinary machine of a uniformly selected N-vertex graph that satisfies the following property: For all but at most an $\epsilon(N) \stackrel{\text{def}}{=} 1/p(\log_2 N)$ fraction of the disjoint set-pairs $(L, R) \subseteq [N] \times [N]$ it holds that the bipartite graph induced by (L, R) has a matching of size $(1 - \epsilon(N)) \cdot \min(|L|, |R|)$.*

As in Theorem 6.8, the implementation is straightforward, and the issue is analyzing it.

Proof: Observe that almost all relevant set-pairs satisfy $|L| \approx |R| \approx N/3$, and so we focus on these pairs. It can still be shown that in a random graph, with overwhelmingly high probability, all the corresponding bipartite graphs (induced by pairs (L, R) as above) have a sufficiently large matching. However, this will not hold if we only consider matchings that *conform with the small bipartite graphs B_i's*, where the B_i's are as in the proof of Theorem 6.8. Still, with overwhelmingly high probability, almost all the bipartite graphs induced by pairs (L, R) as above will have a sufficiently large matching that *does conform* with the small bipartite graphs B_i's. Thus, for $\epsilon = \epsilon(N)$, the distinguisher just selects $O(1/\epsilon)$ different i's, and for each such i tests the size of the maximal matching for $O(1/\epsilon)$ random (L, R)'s. Needless to say, the distinguisher does not select such huge sets, but rather selects their projection on P_i. That is, for each such i (and each attempt), the distinguisher selects a random pair of disjoint sets $(L_i, R_i) \subset P_i \times P_i$. ■

Digest: An interesting aspect regarding the proofs of Theorems 6.8 and 6.9 is that in these cases, with overwhelmingly high probability, a random object in the specification (S, n) has stronger properties that those of arbitrary objects in (S, n). This fact makes it easier to distinguish a random object in (S, n) from an object not in (S, n) (than to distinguish an arbitrary object in (S, n) from an object not in (S, n)). For example, with overwhelmingly high probability, a random graph has larger connectivity than required in Theorem 6.8 and this connectivity is achieved via very short paths (rather than arbitrary ones). This fact enables to distinguish (S, n) from an implementation that lacks sufficiently large connectivity.

A different perspective: The proofs of Theorems 6.8 and 6.9 actually establish that, for the corresponding specifications, the almost-truthfulness of an implementation follows from its computational indistinguishability (w.r.t the specification).[33] An interesting research project is to *characterize the class of specifications for which the foregoing implication holds*; that is, characterize the class of specifications that satisfy Condition 1 in the following Theorem 6.10. Clearly, any pseudo-implementation of such a specification is almost-truthful, and Theorem 6.10 just asserts that having a pseudo-implementation by an oracle machine suffices (provided one-way functions exist):

Theorem 6.10 *Suppose that S is a specification for which the following two conditions hold.*

1. *Every pseudo-implementation of S is almost-truthful to S. In fact, it suffices that this condition holds with respect to implementations by an ordinary probabilistic polynomial-time machines.*
2. *S has an almost-truthful pseudo-implementation* by an oracle machine *that has access to a random oracle.*

[33] That is, these proofs establish the first condition in the following Theorem 6.10, whereas the second condition is established by the straightforward construction of a random graph.

Then, assuming the existence of one-way function, S has an almost-truthful pseudo-implementation by an ordinary probabilistic polynomial-time machine.

Proof: Let I be the implementation guaranteed by Condition 2, and let I' be the implementation derived from I by replacing the random oracle with a pseudorandom function. Then, I' is a pseudo-implementation of S. Using Condition 1, it follows that I' is almost-truthful to S. ■

7 Supporting Complex Queries regarding Random Graphs

In this section we provide truthful implementations of random graph while supporting complex queries, in addition to the standard adjacency queries. Specifically, we consider three types of complex but natural queries (see Proposition 7.1, and Theorems 7.2 and 7.3, respectively). The graph model is as in Section 6, and as in Section 6.1 we present our (truthful) implementations in terms of oracle machines. Let us start with a simple example.

Proposition 7.1 (distance queries and shortest path queries): *There exists a truthful close-implementation by an oracle machine of the following specification. The specifying machine selects uniformly an N-vertex graph and answers distance queries regarding any pair of vertices. Furthermore, there exists a truthful close-implementation of the related specification that returns a uniformly distributed path of shortest length.*

Proof: Consider the property of having diameter at most 2. This property satisfies Condition C (e.g., by an N-vertex star). Thus, using Theorem 6.2, we obtain a close-implementation of a random graph, while our implementation always produces a graph having diameter at most 2 (or rather exactly 2). Now, we answer the query (u,v) by 1 if the edge (u,v) is in the graph, and by 2 otherwise. For the furthermore-part, we add $\sqrt{N}$ such stars, and serve queries regarding paths of length 2 by using the center of one of these stars (which is selected by applying an independent random function to the query pair). ■

The foregoing example is not very impressive because the user could have served the distance-queries in the same way (by only using adjacency queries to the standard implementation of a random graph). (A random shortest path could have also been found by using the standard implementation.) The only advantage of Proposition 7.1 is that it provides a truthful implementation of the distance-queries (rather than merely an almost-truthful one obtained via the trivial implementation). A more impressive example follows.

Serving log-sized clique queries. Recall that a random N-vertex graph is likely to have many $(\log_2 N)$-vertex cliques that include each of the vertices of the graph, whereas it seems hard to find such cliques (where in *hard* we mean unlikely to achieve in time poly($\log N$), and not merely in time poly(N)). Below we provide an implementation of a service that answers queries of the form $v \in [N]$ with a log-sized clique containing the vertex v.

Theorem 7.2 *There exists a truthful close-implementation of the following specification. The specifying machine selects uniformly an N-vertex graph and, in addition to the standard adjacency queries, answers* (Log-Clique) queries *of the form v by providing a random $\lceil \log_2 N \rceil$-vertex clique that contains v (and a special symbol if no such clique exists).*

Proof: Let $\ell = \lceil \log_2 N \rceil - 1$ and consider a simple partition of $[N]$ to $T = \lceil N/\ell \rceil$ subsets, $S_1, ..., S_T$, such that $|S_i| = \ell$ for $i = 1, ..., T-1$ (e.g., $S_i = \{(i-1)\ell + j : j = 1, ..., \ell\}$). Use the oracle to closely-implement a random graph, $G' = ([N], E')$, as well as a random *onto* function[34] $f : [N] \to [T]$ and a random invertible permutation $\pi : [N] \to [N]$ (as in Theorem 2.13). The graph we implement will consist of the union of G' with N cliques, where the i-th clique resides on the vertex set $\{i\} \cup \{\pi(j) : j \in S_{f(i)}\}$. The Log-Clique queries are served in the obvious manner; that is, query v is answered with $\{v\} \cup \{\pi(u) : u \in S_{f(v)}\}$. Indeed,

[34] Such a function can be obtained by combining the identity function over $[T]$ with a random function $f' : \{T+1, ..., N\} \to [T]$, and randomly permuting the *domain* of the resulting function.

for simplicity, we ignore the unlikely case that $v \in \{\pi(u) : u \in S_{f(v)}\}$; this can be redeemed by modifying the implementation as discussed at the end of the proof.

Implementing the adjacency queries is slightly more tricky. The query (u, v) is answered by 1 if and only if either $(u, v) \in E$ or u and v reside in one of the N's cliques we added. The latter case may happen if and only if one of the following subcases holds:

1. Either $u \in \{\pi(w) : w \in S_{f(v)}\}$ or $v \in \{\pi(w) : w \in S_{f(u)}\}$; that is, either $\pi^{-1}(u) \in S_{f(v)}$ or $\pi^{-1}(v) \in S_{f(u)}$. Each of these conditions is easy to check by invoking f and π^{-1}.
2. There exists an x such that $u, v \in \{\pi(w) : w \in S_{f(x)}\}$, which means that $\pi^{-1}(u), \pi^{-1}(v) \in S_{f(x)}$. Equivalently, recalling that f is onto, we may check whether there exists a y such that $\pi^{-1}(u), \pi^{-1}(v) \in S_y$, which in turn is easy to determine using the simple structure of the sets S_y's (i.e., we merely test whether or not $\lceil \pi^{-1}(u)/\ell \rceil = \lceil \pi^{-1}(v)/\ell \rceil$).

Thus, our implementation is truthful to the specification. To see that it is a close-implementation of the specification, observe first that it is unlikely that two different Log-Clique queries are "served" by the same clique (because this means forming a collision under f). Conditioned on this rare event not occurring, the Log-Clique queries are served by disjoint random cliques, which is what would essentially happen in a random graph (provided that at most poly($\log N$) queries are made). Finally, it is unlikely that the answers to the adjacency queries that are not determined by prior Log-Clique queries be affected by the sparse sub-graph (of N small cliques) that we inserted under a random permutation.

Finally, we address the problem ignored above (i.e., the rare case when the query v is in the corresponding set $\{\pi(u) : u \in S_{f(v)}\}$). We modify the foregoing implementation by setting $\ell = \lceil \log_2 N \rceil$ (rather than $\ell = \lceil \log_2 N \rceil - 1$), and using corresponding sets of size ℓ. Note that, under this modification, for most vertices v, the set $\{v\} \cup \{\pi(u) : u \in S_{f(v)}\}$ has size $\ell + 1$ (whereas for few vertices v this set has size ℓ). Thus, in the modified implementation, a query v is answered with a random ℓ-subset of $\{v\} \cup \{\pi(u) : u \in S_{f(v)}\}$ that contains v (i.e., we use another random function $g : [N] \to [\ell]$ that indicates which element of $\{\pi(u) : u \in S_{f(v)}\}$ to drop in the case that $v \notin \{\pi(u) : u \in S_{f(v)}\}$). The theorem follows. ■

Another example: queries regarding a fixed Hamiltonian cycle. We consider the implementation of a random graph along with answering queries regarding a fixed random Hamiltonian cycle in it, where such a cycle exists with overwhelmingly high probability. Specifically, we consider queries of the form *what is the distance between two vertices on the cycle.*

Theorem 7.3 *There exists a truthful close-implementation of the following specification. The specifying machine selects uniformly an N-vertex Hamiltonian graph G, and uniformly selects a (directed) Hamiltonian cycle in G, which in turn defines a cyclic permutation $\sigma : [N] \to [N]$. In addition to the standard adjacency queries, the specification answers* travel queries *of the form* (trav, v, t) *by providing $\sigma^t(v)$, and* distance queries *of the form* (dist, v, w) *by providing the smallest $t \geq 0$ such that $w = \sigma^t(v)$.*

We stress that the implementation must answer each possible query in time polynomial in the vertex name (which may be logarithmic in the distance t).

Proof: It will be convenient to use the vertex set $V = \{0, 1, ..., N-1\}$ (instead of $[N]$). We use the random oracle to closely-implement a random graph $G' = (V, E')$ as well as a random permutation $\pi : V \to V$ along with its inverse. We define a graph $G = (V, E)$ by $E \stackrel{\text{def}}{=} E' \cup C$, where $C = \{(\pi(i), \pi(i + 1 \bmod N)) : i \in V\}$, and use C to answer the special (Hamiltonian) queries. That is, we answer the query (trav, v, t) by $\pi(\pi^{-1}(v) + t \bmod N)$, and the query (dist, v, w) by $\pi^{-1}(w) - \pi^{-1}(v) \bmod N$. The standard adjacency query (u, v) is answered by 1 if and only if either $(u, v) \in E$ or $\pi^{-1}(u) \equiv \pi^{-1}(v) \pm 1 \pmod{N}$. (Indeed, the above construction is reminiscent of the "fast-forward" construction of [38] (stated in Theorem 2.14).)

To see that the above truthful implementation is statistically-indistinguishable from the specification, we use the following three observations:

1. If a (labeled) graph appears in the specification (resp., in the implementation) then all is (labeled) isomorphic copies appear in it. Consequently, for any fixed Hamiltonian cycle, the set of Hamiltonian graphs in which this cycle has been selected in the specification (resp., in the implementation) is isomorphic to the set of Hamiltonian graphs in which any other fixed Hamiltonian cycle has been selected. Thus, we may consider the conditional distribution induced on the specification (resp., on the implementation) by fixing any such Hamiltonian cycle.
2. Conditioned on any fixed Hamiltonian cycle being selected in the implementation, the rest of the graph selected by the implementation is truly random.
3. Conditioned on any fixed Hamiltonian cycle being selected in the specification, the rest of the graph selected by the specification is indistinguishable from a random graph. The proof of this assertion is similar to the proof of Lemma 6.3. The key point is proving that, conditioned on a specific Hamiltonian cycle being selected, the (rest of the) graph selected by the specification has sufficiently high entropy. Note that here we refer to the entropy of the remaining $\binom{N}{2} - N$ edges, and that the vertex pairs are not all identical but rather fall into categories depending on their distance as measured on the selected Hamiltonian cycle. We need to show that a random vertex-pair in *each* of these categories has a sufficiently high (conditional) entropy. Thus, this observation requires a careful proof to be presented next.

Indeed, the foregoing discussion suggests that we may give the entire Hamiltonian cycle to the machine that inspects the rest of the graph (in an attempt to distinguish the implementation from the specification). Thus, we assume, without loss of generality, that this machine makes no adjacency queries regarding edges that participate in the cycle. The first observation says that we may consider any fixed cycle, and the second observation says that a machine that inspects the rest of the implementation (i.e., the graph that is constructed by the implementation) sees truly random edges. The third observation, proved below, asserts that making a few queries to the rest of the conditional space of the specification, yields answers that also look random.

We consider the conditional distribution of the rest of the graph selected by the specification, given that a specific Hamiltonian cycle was selected. (Indeed, we ignore the negligible (in N) probability that the graph selected by the specification is not Hamiltonian.) Essentially, the argument proceeds as follows. First, we note that (by Bayes' Law) the conditional probability that a specific graph is selected is inversely proportional to the number of Hamiltonian cycles in that graph. Next, using known results on the concentration of the latter number in random graphs (see, e.g., [28, Thm. 4]), we infer that in all but an N^{-2} fraction of the N-vertex graphs the number of Hamiltonian cycles is at least an $\exp(-2(\ln N)^{1/2}) > N^{-1}$ fraction of its expected number. Thus, we conclude that the conditional entropy of the selected graph (conditioned on the selected cycle) is $\binom{N}{2} - N - o(N)$. Details follow.

For $T = \binom{N}{2}$, let $X = X_1 \cdots X_T$ denote the graph selected by the specification, and $Y(G)$ denote the Hamiltonian cycle selected (by the specification) given that the graph G was selected. Let $\#_{\mathtt{HC}}(G)$ denote the number of Hamiltonian cycles in the graph G, where cyclic shifts and transpositions of cycles are counted as if they were different cycles (and so the number of Hamiltonian cycles in an N-clique is $N!$). Thus, $\mathbf{E}(\#_{\mathtt{HC}}(X)) = 2^{-N} \cdot (N!)$. An N-vertex graph G is called good if $\#_{\mathtt{HC}}(G) > 2^{-N} \cdot ((N-1)!)$, and $\mathcal{G}$ denotes the set of good N-vertex graphs. For a Hamiltonian cycle C, we denote by $\mathcal{G}(C)$ the set of graphs in $\mathcal{G}$ that contain the cycle C. Then, it holds that

$$\begin{aligned}
\mathbf{H}(X|Y(X)=C) &\geq \sum_{G\in\mathcal{G}(C)} \mathbf{Pr}[X=G|Y(X)=C] \cdot \log_2(1/\mathbf{Pr}[X=G|Y(X)=C]) \\
&\geq (1-N^{-2}) \cdot \min_{G\in\mathcal{G}(C)} \{-\log_2(\mathbf{Pr}[X=G|Y(X)=C])\} \\
&= (1-N^{-2}) \cdot \min_{G\in\mathcal{G}(C)} \left\{ \begin{array}{l} \log_2(\mathbf{Pr}[Y(X)=C]) \\ -\log_2(\mathbf{Pr}[Y(X)=C|X=G]) \\ -\log_2(\mathbf{Pr}[X=G]) \end{array} \right\} \\
&= (1-N^{-2}) \cdot \min_{G\in\mathcal{G}(C)} \left\{ \log_2(1/N!) + \log_2(\#_{\mathtt{HC}}(G)) + \binom{N}{2} \right\}
\end{aligned}$$

Using the fact that G is good (i.e., $G \in \mathcal{G}(C)$), it follows that $\log_2(\#_{\mathtt{HC}}(G)) > \log_2(2^{-N} \cdot ((N-1)!))$, which in turn equals $\log_2(N!) - N - \log_2 N$. We thus get,

$$\mathbf{H}(X|Y(X) = C) \; > \; (1 - N^{-2}) \cdot \left(\binom{N}{2} - N - \log_2 N \right) \tag{3}$$

Recall that the condition $Y(X) = C$ determines N vertex-pairs in X, and so the entropy of the remaining $T' = \binom{N}{2} - N$ pairs is at least $T' - \log_2 N$. Partitioning these (undetermined) pairs according to their distances in C, we conclude that the entropy of the $N/2$ pairs in each such distance-class is at least $(N/2) - \log_2 N$. (Indeed, the distance class of undetermined pairs do not contain distance 1 (or $N-1$), which correspond to the forced cycle-edges.) We stress that our analysis holds even if the machine inspecting the graph is given the Hamiltonian cycle for free. This machine may select the induced subgraph that it wants to inspect, but this selection is determined up to a shifting of all vertices (i.e., a rotation of the cycle). This randomization suffices for concluding that the expected entropy of the inspected subgraph (which may not include cycle edges) is at least $(1 - ((2\log_2 N)/N)) \cdot \binom{t}{2}$, where t is the number of vertices in the subgraph. As in the proof of Lemma 6.3, this implies that the inspected subgraph is at distance at most $O(\sqrt{((\log_2 N)/N) \cdot \binom{t}{2}}) < t \cdot N^{-(1-o(1))/2}$ from a random t-vertex graph. The theorem follows. ■

8 Random Bounded-Degree Graphs and Global Properties

In this section we consider huge bounded-degree simple graphs, where the vertices are labeled (and there are no self-loops or parallel edges). We consider specifications of various distributions over such graphs, where in all cases the specifying machine responds to neighborhood queries (i.e., the queries correspond to vertices and the answer to query v is the list of all vertices that are adjacent to vertex v).

The first issue that arises is whether we can implement a random bounded-degree graph or alternatively a random regular graph. Things would have been quite simple if we were allowing also non-simple graphs (i.e., having self-loops and parallel edges). For example, a random d-regular N-vertex *non-simple* graph can be implemented by pairing at random the dN possible "ports" of the N vertices. We can avoid self-loops (but not parallel edges) by generating the graph as a union of d perfect matchings of the elements in $[N]$. In both cases, we would get a close-implementation of a random d-regular N-vertex (simple) graph, but parallel edges will still appear with constant probability (and thus this implementation is not truthful w.r.t simple graphs). In order to obtain a random *simple* d-regular N-vertex graph, we need to take an alternative route. The key observation underlying this alternative is captured by the following lemma:

Lemma 8.1 *For $d > 2$, let $G = ([N], E)$ be any d-regular N-vertex graph having girth g. Let G' be obtained by randomly permuting the vertices of G (and presenting the incidence lists in some canonical order). Then, any machine M that queries the graph for the neighborhoods of q vertices of its choice, cannot distinguish G' from a random d-regular N-vertex (simple) graph, except than with probability $O(q^2/(d-1)^{(g-1)/2})$. In the case $d = 2$ and $q < g - 1$, the probability bound can be improved to $O(q^2/N)$.*

Recall that the girth of a graph G is the length of the shortest *simple* cycle in G, and that $(d-1)^{(g-2)/2} < N$ always holds (for a d-regular N-vertex graph of girth g).[35] Note that Lemma 8.1 is quite tight: For example, in the case $d = 2$, for $g \ll \sqrt{N}$, the N-vertex graph G may consist of a collection of g-cycles, and taking a walk of length g in G' (by making $g - 1$ queries) will always detect a cycle G', which allows to distinguish G' from a random 2-regular N-vertex (in which the expected length of a cycle going through any vertex is $\Omega(N)$). In the case $d \geq 3$, the graph G may consist of connected components, each of size $(d-1)^g \ll N$, and taking a random walk of length $(d-1)^{g/2}$ in G' is likely to visit some vertex twice, which allows to distinguish G' from a random d-regular N-vertex (in which this event may occur only after $\sqrt{N}$ steps). Below, we will use Lemma 8.1 with the following setting of parameters.

[35] The girth upper-bound (i.e., $g \leq 2 + 2\log_{d-1} N$) follows by considering the (vertex disjoint) paths of length $(g-2)/2$ starting at any fixed vertex. The existence of d-regular N-vertex graphs of girth $\log_{d-1} N$ was shown (non-constructively) in [13].

Corollary 8.2 *For fixed $d > 2$ and $g(N) = \omega(\log\log N)$, let $G = ([N], E)$ be any d-regular N-vertex graph having girth $g(N)$. Let G' be obtained from G as in Lemma 8.1. Then, any machine M that queries the graph for the neighborhoods of* $\mathrm{poly}(\log N)$ *vertices of its choice, cannot distinguish G' from a random d-regular N-vertex (simple) graph, except than with negligible in* $\log N$ *probability. The claim holds also in the case that $d = 2$ and $g(N) = (\log N)^{\omega(1)}$.*

For $d > 2$ the girth can be at most logarithmic, and explicit constructions with logarithmic girth are known for all $d \geq 3$ and a dense set of N's (which is typically related to the set of prime numbers; see, e.g., [35, 27, 33]). For $d = 2$, we may just take the N-cycle or any N-vertex graph consisting of a collection of sufficiently large cycles.

Proof of Lemma 8.1: We bound the distinguishing gap of an oracle machine (which queries either a random d-regular N-vertex graph or the random graph G') as a function of the number of queries it makes. Recall that G' is a random isomorphic copy of G, whereas a random d-regular N-vertex graph may be viewed as a random isomorphic copy of another random d-regular N-vertex graph. Thus, intuitively, the specific labels of queried vertices and the specific labels of the corresponding answers are totally irrelevant: the only thing that matters is whether or not two labels are equal.[36] Equality (between labels) can occur in two cases. The *uninteresting case* is when the machine queries a vertex u that is a neighbor of a previously-queried vertex v and the answer contains (of course) the label of vertex v. (This is uninteresting because the machine, having queried v before, already knows that v is a neighbor of u.) The interesting case is that the machine queries a vertex and the answer contains the label of a vertex v that was not queried before but has already appeared in the answer to a different query. An important observation is that, as long as no interesting event occurs, the machine cannot distinguish the two distributions (because in both cases it knows the same subgraph, which is a forest). Thus, the analysis amounts to bounding the probability that an interesting event occurs, when we make q queries.

Let us consider first what happens when we query a random d-regular N-vertex (simple) graph. We may think of an imaginary process that constructs the graph on-the-fly such that the neighbors of vertex v are selected only in response to the query v (cf, e.g., the proof of [22, Thm. 7.1]). This selection is done at random according to the conditional distribution that is consistent with the partial graph determined so far. It is easy to see that the probability that an interesting event occurs in the i-th query is at most $(i-1)d/(dN-(i-1)d)$, and so the probability for such an event occurring in q queries is at most q^2/N.

The more challenging part is to analysis what happens when we query the graph G. (Recall that we have already reduced the analysis to a model in which we ignore the specific labels, but rather only compare them, and analogously we cannot query a specific new vertex but rather only query either a random new vertex or a vertex that has appeared in some answer.)[37] To illustrate the issues at hand, consider first the case that $d = 2$ (where G consists of a set of cycles, each of length at least g). In this case, we have the option of either to proceed along a path that is part of a cycle (i.e., query for the neighbors of the an end-point of a currently known path) or to query for a random new vertex. Assuming that we make less than $g-1$ queries, we can never cause an interesting event by going along a path (because an interesting event may occur in this case only if we go around the entire cycle, which requires at least $g-1$ queries). The only other possibility to encounter an interesting event is by having two paths (possibly each of length 1) collide. But the probability for such an event is bounded by q^2/N, where q is the number of queries that we make.[38]

[36] Essentially, the machine cannot determine which vertex it queries; all that it actually decides is whether to query a specific vertex that has appeared in previous answers or to query a new vertex (which may be viewed as randomly selected). (Formally, a specific new label indicated by the querying machine is mapped by the random permutation to a new random vertex.) Similarly, the labels of the vertices given as answer do not matter, all that matters is whether or not these vertices have appeared in the answers to previous queries (or as previous queries). (Again, formally, the new vertices supplied in the answer are assigned, by the random permutation, new random labels.)

[37] Thus, we may consider querying G itself (rather than querying G').

[38] Using a union bound over all query pairs, we bound the probability that the i^{th} query collides with the j-th query. Each of these two queries is obtained by a path of fixed length starting from a uniformly and distributed vertex (which was new at the time). Thus, these two queries are almost uniformly and independently distributed (in $[N]$), and the probability that they are neighbors is at most $1/(N-q)$.

We now turn to the more interesting case of $d > 2$. As in case $d = 2$, taking a walk of length $g - 2$ from any vertex will not yield anything useful. However, in this case, we may afford to take longer walks (because q may be much larger than g). Still, we will prove that, in this case, with probability at least $1 - q^2 \cdot (d-1)^{-(g-3)/2}$, the uncovered subgraph is a forest. The proof relies both on the girth lower-bound of G and on a sufficiently-good rapid-mixing property (which follows from the girth lower-bound). We bound the probability that a cycle is closed in the current forest by the probability that two vertices in the forest are connected by a non-tree edge, where the probability is taken over the possible random vertices returned in response to a new-vertex request and over the random order in which neighbors of a query-vertex are provided. Indeed, a key observation is that when we query a vertex that has appeared in some answer, we may think that this vertex is selected at random among the unqueried vertices appearing in that answer.[39] Taking a union bound on all possible $\binom{q}{2}$ vertex pairs (i.e., those in the forest), we bound the probability that either two ends of a discovered path (in one tree) or two vertices in different current trees are connected by an edge. (In both cases, these vertices are actually leaves.)

We consider each of these two cases separately: In the latter case (i.e., leaves in different trees), the two vertices (which are not connected in the currently uncovered subgraph) are uniformly distributed in G, and thus the probability that they are connected is essentially d/N. The situation here is essentially as analyzed in the case $d = 2$: we have two paths, each initiated at a random (new at the time) vertex, leading to the leaves in question, and thus the latter are almost uniformly and independently distributed.

Turning to the former case (i.e., endpoints of a path in a tree), we use the girth hypothesis to infer that this path must have length at least $g - 1$ (or else its endpoint are definitely not connected). However, the machine that discovered this path actually took a random walk (possibly to two directions) starting from one vertex, because we may assume that this is the first time in which two vertices in the current forest are connected by a current non-tree edge. We also use the hypothesis that our exploration of the path (i.e., queries regarding vertices that appeared in previous answers) is actually random (i.e., we effectively extend the current end-point of the path by a uniformly selected neighbor of that end-point). Now, the end-point of such a path cannot hit any specific vertex with probability greater than $\nu \stackrel{\text{def}}{=} (d-1)^{-(g-1)/2}$, because after $(g-1)/2$ steps the end-point must be uniformly distributed over the $(d-1)^{(g-1)/2}$ leaves of the tree rooted at the start vertex (and the max-norm of a distribution cannot increase by additional random steps). Fixing the closest (to the start vertex) end-point, it follows that the probability that the other end-point hits the neighbor-set of the first end-point is at most $d \cdot \nu = O((d-1)^{-(g-1)/2})$. To summarize, the probability that an interesting event occurs while making q queries is $O(q^2 \cdot (d-1)^{-(g-1)/2})$. The lemma follows. ■

Implementing random bounded-degree simple graphs: We now turn back to the initial problem of implementing random bounded-degree (resp., regular) simple graphs.

Proposition 8.3 *For every constant $d > 2$, there exist truthful close-implementations of the following two specifications:*

1. A random graph of maximum degree d: *For size parameter N, the specification selects uniformly a graph G among the set of N-vertex simple graphs having maximum degree d. On query $v \in [N]$, the machine answers with the list of neighbors of vertex v in G.*
2. A random d-regular graph: *For size parameter N, the specification selects uniformly a graph G among the set of N-vertex d-regular simple graphs, and answers queries as in Part 1.*

Proof: We start with Part 2. This part should follow by Corollary 8.2, provided that we can implement a random isomorphic copy of a d-regular N-vertex graph of sufficiently large girth. This requires an explicit

[39] That is, the correspondence between the new place-holders in the answer and the new real neighbors of the queried vertex is random. Formally, we may define the interaction with the graph such that at each point only the internal nodes of the currently revealed forest are assigned a serial number. Possible queries may be either for a new random vertex (assigned the next serial number and typically initiating a new tree in the forest) or for a random leaf of a specific internal vertex (which typically extends the corresponding tree and turns one of these leaves to an internal vertex with $d-1$ new leaves).

construction of the latter graph as well as a close-implementation of a random permutation and its inverse (as provided by Theorem 2.13). Specifically, let G_N be the fixed graph, and π the random relabeling of its vertices. We answer query v, by first determining the preimage of v in G_N (i.e., $\pi^{-1}(v)$), next find its neighbors (using the explicitness of the construction of G_N), and finally return their images under π. Indeed, this process depends on the ability to provide explicit constructions of adequate d-regular N-vertex graphs (i.e., G_N's). This is trivial in the case $d = 2$ (e.g., by the N-cycle). For other values of $d \geq 3$, adequate constructions can be obtained from [35, 27, 33, 30] (possibly by dropping several (easily identified) perfect matchings from the graph). These construction apply for a dense set of N's (which are typically of the form $p(p-1)^2$ for any prime p), but we can obtain other sizes by combining many such graphs (note that we are not even required to give a connected graph, let alone a good expander).

We now turn to Part 1. We first note that most graphs of maximum degree d have $(1-o(1)) \cdot dN/2$ edges. Furthermore, for $T = \Theta(\sqrt{dN})$ and $D = O(\sqrt{dN})$, all but a negligible (in N) fraction of the graphs have $(dN/2) - T \pm D$ edges. Thus, a random N-vertex graph of degree bound d is statistically-indistinguishable from a random d-regular graph with N vertices, because the former may be viewed as resulting from omitting a small number (i.e., $T + D = O(\sqrt{N})$) of edges from a random d-regular graph with N vertices. ■

A general result: The proof of Proposition 8.3 actually yields a truthful close-implementation of several other specifications. Consider, for example, the generation of random connected d-regular graphs, for $d \geq 3$. Since the explicit constructions of d-regular graphs are connected (and their modifications can easily made connected), applying Corollary 8.2 will do. (Indeed, we also use the fact that, with overwhelmingly high probability, a random d-regular graph is connected.) More generally, we have:

Theorem 8.4 *Let $d \geq 2$ be fixed and Π be a graph property that satisfies the following two conditions:*

1. *The probability that Property Π is not satisfied by a uniformly chosen d-regular N-vertex graph is negligible in $\log N$.*
2. *Property Π is satisfied by a family of strongly-constructible d-regular N-vertex graphs having girth $\omega(\log\log N)$ if $d > 2$ and girth $(\log N)^{\omega(1)}$ if $d = 2$.*

Then, there exists a truthful close-implementation (by an oracle machine) of a uniformly distributed d-regular N-vertex graph that satisfies property Π.

We note that Condition 1 may be relaxed. It suffices to require that a random d-regular graph and a random d-regular graph having Property Π are statistically-indistinguishable (by a machine that makes poly-logarithmically many queries). In particular, a random 2-regular graph and a uniformly distributed *connected* 2-regular graph are statistically-indistinguishable, and thus we can provide a truthful close-implementation of the latter specification. We mention that Theorem 8.4 yields truthful close-implementations to random d-regular graphs that are required to be Hamiltonian, Bipartite, have logarithmic girth, etc.

9 Complex Queries regarding Length-Preserving Functions

In this section we consider specifications that refer to a generic random function, but support complex queries regarding such functions. That is, we consider answer various queries regarding a random function $f : \{0,1\}^n \to \{0,1\}^n$, in addition to the standard evaluation queries. The first type of complex queries that we handle are iterated-evaluation queries, where the number of iterations may be super-polynomial in the length of the input (and thus cannot be implemented in a straightforward manner).

Theorem 9.1 (iterated-evaluation queries to a random mapping): *For every positive polynomial p, there exists a truthful close-implementation of the following specification. The specifying machine, uniformly selects a function $f : \{0,1\}^n \to \{0,1\}^n$, and answers queries of the form (x, m), where $x \in \{0,1\}^n$ and $m \in [2^{p(n)}]$, with the value $f^m(x)$ (i.e., f iterated m times on x).*

Proof: It will be convenient to associate $\{0,1\}^n$ with $\{0,1,...,N-1\}$, where $N=2^n$. As a warm-up, consider an implementation by a random N-cycle; that is, using a random 1-1 mapping $\pi:\{0,...,N-1\}\to\{0,1\}^n$, define $f(x)=\pi(\pi^{-1}(x)+1 \bmod N)$, and answer the query (x,m) by $f^m(x)=\pi(\pi^{-1}(x)+m \bmod N)$. (Indeed, this construction is reminiscent of the "fast-forward" construction of [38] (stated in Theorem 2.14).) The only thing that goes wrong with this construction is that we know the cycle length of f (i.e., it is always N), and thus can distinguish f from a random function by any query of the form $(\cdot,N)$. Thus, we modify the construction so to obtain a function f with unknown cycle lengths. A simple way of doing this is to use two cycles, while randomly selecting the length of the first cycle. That is, select M uniformly in $[N]$, and let

$$f(x) \stackrel{\rm def}{=} \begin{cases} \pi(\pi^{-1}(x)+1 \bmod M) & \text{if } \pi^{-1}(x)\in\{0,...,M-1\} \\ \pi(\pi^{-1}(x)+1) & \text{if } \pi^{-1}(x)\in\{M,...,N-2\} \\ \pi(M) & \text{otherwise (i.e., } \pi^{-1}(x)=N-1) \end{cases}$$

We could have tried to select f such that its cycle structure is distributed as in case of a random function, but we did not bother to do so. Nevertheless, we prove that any machine that makes q queries cannot distinguish f from a random function with probability better than $\mathrm{poly}(n)\cdot q^2/2^{\Omega(n)}$. Actually, in order to facilitate the analysis, we select M uniformly in $\{(N/3),...,(2N/3)\}$.

We turn to prove that the foregoing (truthful) implementation is statistically-indistinguishable from the specification. As in the proof of Lemma 8.1, we may disregard the *actual* values of queries and answers (in the querying process), and merely refer to whether these values are equal or not. We also assume, without loss of generality, that the querying machine makes no redundant queries (e.g., if the machine "knows" that $y=f^k(x)$ and $z=f^\ell(y)$ then it refrains from making the query $(x,k+\ell)$, which would have been answered by $z=f^{k+\ell}(x)$). Thus, at any point in time, the querying machine knows of a few chains, each having the form $(x,f^{k_1}(x),f^{k_2}(x),...,f^{k_t}(x))$, for some known $x\in\{0,1\}^n$ and $k_1<k_2<\cdots<k_t$. Typically, the elements in each chain are distinct, and no element appears in two chains. In fact, as long as this typical case holds, there is no difference between querying the specification versus querying the implementation. Thus, we have to upper bound the probability that an untypical event occurs (i.e., a query is answered by an element that already appears on one of the chains, although the query was not redundant).

Let us first consider the case that f is constructed as in the implementation. For the i-th non-redundant query, denoted (x,k), we consider three cases:

Case 1: x does not reside on any chain. The probability that $f^k(x)$ hits a known element is at most $(i-1)/(N-(i-1))$, because x is uniformly distributed among the $N-(i-1)$ unknown elements. (Since f is 1-1, it follows that $f^k(x)$ is uniformly distributed over a set of $N-(i-1)$ elements.)

Case 2: x resides on one chain and $f^k(x)$ hits another chain. We show that the probability to hit an element of another chain (which must belong to the same cycle) is $(i-1)/(N'-(i-1)^2)$, where $N'\geq N/3$ is the number of vertices on the cycle (on which x reside). The reason is that chains residing on the same cycle may be thought of as having a random relative shift (which must be such that avoids any collisions of the up-to $i-1$ known vertices). For $i<\sqrt{N/2}$, we obtain a probability bound of $i/\Omega(N)$.

Case 3: x resides on some chain and $f^k(x)$ hits the same chain. Without loss of generality, suppose that $f^k(x)=x$. For this to happen, the length N' of the cycle (on which x reside) must divide k. We upper-bound the probability that all prime factors of N' are prime factors of k.

Recall that N' is uniformly selected in $[(N/3),(2N/3)]$, and let $P=P_k$ denote the set of prime factors of k. Note that for some constant c, it holds that $|P|<n^{c-1}$, because by the hypothesis $k\in[2^{\mathrm{poly}(n)}]$. We upper-bound the number of integers in $[N]$ that have all prime factors in P by upper-bounding, for every $t\in[n]$, the product of the number of integers in $[2^t]$ with all prime factors in $P'\stackrel{\rm def}{=}\{p\in P: p<n^c\}$ and the number of $(n-t)$-bit integers with all prime factors in $P''\stackrel{\rm def}{=}P\setminus P'$. For $t>n/\log n$, the size of the first set can be upper-bounded by the number of n^c-smooth numbers in $[2^t]$, which in turn is upper-bounded by $2^{t-(t/c)+o(t)}=2^{(1-(1/c))\cdot t+o(t)}$.[40] The size of the second set

[40] An integer is called y-smooth if all its prime factors are smaller that y. The fraction of y-smooth integers in $[x]$ is upper-bounded by $u^{-u+o(u)}$, where $u=(\log x)/(\log y)$; see, [10]. Thus, in case $t>n/\log n$, the fraction of n^c-smooth integers in $[2^t]$ is upper-bounded by $2^{-(1-o(1))\cdot(t/(c\log_2 n))\cdot\log_2 t}=2^{-(1-o(1))t/c}$.

is upper-bounded by $|P''|^{(n-t)/(c\log n)} < 2^{(1-(1/c))\cdot(n-t)}$, where the inequality uses $|P''| < n^{c-1}$. Thus, we upper-bound the probability that an uniformly chosen integer in $[(N/3),(2N/3)]$ has all prime factors in P by

$$\begin{aligned}
& \sum_{t=1}^{n/\log n} 1 \cdot 2^{-(1/c)\cdot(n-t)} + \sum_{t=(n/\log n)+1}^{n} 2^{-(1/c)\cdot t+o(t)} \cdot 2^{-(1/c)\cdot(n-t)} \\
= & \sum_{t=1}^{n/\log n} 2^{-(1/c)\cdot(n-t)} + \sum_{t=(n/\log n)+1}^{n} 2^{-(1/c)\cdot n+o(t)} \\
= & \ 2^{-(n/c)+o(n)}
\end{aligned}$$

Hence, the probability of a collision in the current case is upper-bounded by $N^{-1/(c+1)}$.

We conclude the probability that we form a collision in q queries (to the implementation) is at most $O(q^2/N)+q\cdot N^{-1/(c+1)} < q^2\cdot N^{-\Omega(1)}$.

We now turn to the case that f is a random function (as in the specification). Suppose that we make the non-redundant query (x,k). We wish to upper-bound the probability that $f^k(x)=y$, for some fixed y (which is on one of the chains). It is well-known that the expected number of ancestors of y under a random f is $\Theta(\sqrt{N})$; see, e.g., Theorem 33 in [8, Ch. XIV]. Thus, $\mathbf{Pr}_f[|\cup_{i\geq 1} f^{-i}(y)| > N^{3/4}] = O(N^{-1/4})$, and it follows that $\mathbf{Pr}_f[f^k(x)=y] < N^{-1/4}+O(N^{-1/4})$, for any fixed (x,k) and y. (Indeed, it seems that this is a gross over-estimate, but it suffices for our purposes.) It follows that the probability that we form a collision in q queries to the specification is at most $O(q^2/N^{1/4})$. ∎

Comment: The proof of Theorem 9.1 can be easily adapted so to provide a truthful close-implementation of a random permutation with iterated-evaluation and iterated-inverse queries. That is, we refer to a specifying machine that uniformly selects a permutation $f:\{0,1\}^n \to \{0,1\}^n$, and answers queries of the form (x,m), where $x\in\{0,1\}^n$ and $m\in[\pm 2^{\mathrm{poly}(n)}]$, with the value $f^m(x)$. The implementation is exactly the one used in the foregoing proof of Theorem 9.1, and thus we should only analyze the a probability of collision when making (non-redundant) queries to a random permutation π. For any fixed (x,k) and y, the probability that $\pi^k(x)=y$ equals the probability that x and y resides on the same cycle of the permutation π and that their distance on this cycle equals $k \bmod \ell$, where ℓ is the length of this cycle. In the case that $x\neq y$, the said event occurs with probability at most $(N-1)^{-1}$, because we may think of first selecting a cycle-structure (and later embedding x and y on it). In the other case (i.e., $x=y$), we note that the probability that $\pi^k(x)=x$ equals the probability that ℓ divides k, whereas ℓ is distributed uniformly over $[N]$ (i.e., for every $i\in[N]$, the probability that $\ell=i$ equals $1/N$). We mention that an alternative implementation of a random permutation supporting iterated-evaluation (and iterated-inverse) queries was suggested independently by Tsaban [39]. Interestingly, his implementation works by selecting a cycle structure with distribution that is statistically-close to that in a random permutation (and using a set of cycles of corresponding lengths, rather than always using two cycles as we do).

Preimage queries to a random mapping: We turn back to random length preserving functions. Such a random function $f:\{0,1\}^n \to \{0,1\}^n$ is highly unlikely to be 1-1, still the set of preimages of an element under the function is well-defined (i.e., $f^{-1}(y)=\{x : f(x)=y\}$). Indeed, this set may be empty, be a singleton or contain more than one preimage. Furthermore, with overwhelmingly high probability, all these sets are of size at most n. The corresponding "inverse" queries are thus natural to consider.

Theorem 9.2 *There exists a truthful close-implementation of the following specification. The specifying machine, uniformly selects a function $f:\{0,1\}^n \to \{0,1\}^n$, and, in addition to the standard evaluation queries, answers the inverse-query $y\in\{0,1\}^n$ with the value $f^{-1}(y)$.*

Proof: We start with a truthful implementation that is not statistically-indistinguishable from the specification, but is "close to being so" and does present our main idea. For $\ell = O(\log n)$ (to be determined), we consider an implementation that uses the oracle in order to define two permutations π_1 and π_2 over $\{0,1\}^n$ (along with their inverses) as well as a random function $g:\{0,1\}^n\to\{0,1\}^\ell$. It is instructive to note that g induces a collection of random independent functions $g_\alpha:\{0,1\}^\ell\to\{0,1\}^\ell$ such that $g_\alpha(\beta)=g(\alpha\beta)$, and that each g_α induces a random function on the corresponding set $S_\alpha \stackrel{\rm def}{=} \{\alpha\beta:\beta\in\{0,1\}^\ell\}$ (i.e., mapping $\alpha\beta$ to $\alpha g_\alpha(\beta)$). Letting $\mathsf{pref}_i(z)$ (resp., $\mathsf{suff}_i(z)$) denote the i-bit long prefix of z (resp., suffix of z), we define

$$
\begin{aligned}
f(x) &= \pi_2\left(\mathsf{pref}_{n-\ell}(\pi_1(x))\, g_{\mathsf{pref}_{n-\ell}(\pi_1(x))}(\mathsf{suff}_\ell(\pi_1(x)))\right) \qquad (4)\\
&= \pi_2\left(\mathsf{pref}_{n-\ell}(\pi_1(x))\, g(\pi_1(x))\right).
\end{aligned}
$$

That is, the value of $f(x)$ is obtained by first routing x to a random value $v\leftarrow\pi_1(x)$, which is viewed as a pair $(\alpha,\beta)=(\mathsf{pref}_{n-\ell}(v),\mathsf{suff}_\ell(v))$, next computing the value $w\equiv(\alpha,g_\alpha(\beta))$, and finally routing w to a random $\pi_2(w)$. Indeed, the functions g_α induces collisions within the structured sets S_α, and so the resulting function f is unlikely to be 1-1.

The evaluation queries are answered in a straightforward way (i.e., by evaluating π_1, g and π_2). The inverse-query y is answered by first computing $\alpha\beta=\pi_2^{-1}(y)$, where $|\alpha|=n-\ell$, then computing $R_\alpha(\beta)\stackrel{\rm def}{=}\{\beta':g(\alpha\beta')=\beta\}$ via exhaustive search, and finally setting $f^{-1}(y)=\{\pi_1^{-1}(\alpha\beta'):\beta'\in R_\alpha(\beta)\}$. Indeed, the key point is that, since $\ell=O(\log n)$, we can afford to determine the set $R_\alpha(\beta)$ by going over all possible $\beta'\in\{0,1\}^\ell$ and including β' if and only if $g(\alpha\beta')=\beta$. The random permutation π_1 (resp., π_2) guarantees that it is unlikely to make two evaluation queries (resp., inverse-queries) that are served via the same set S_α (i.e., have the same $(n-\ell)$-bit long prefix under the relevant permutation). It is also unlikely to have a non-obvious "interaction" between these two types of queries (where an obvious interaction is obtained by asking for a preimage of an answer to an evaluation query or vice versa). Thus, the answers to the evaluation queries look random, and the answers to the inverse-queries are almost independent random subsets with sizes that corresponds to the statistics of the collision of 2^ℓ elements (i.e., 2^ℓ balls thrown at random to 2^ℓ cells).

The only thing that is wrong with the foregoing implementation is that the sizes of the preimage-sets correspond to the collision pattern of 2^ℓ balls thrown at random to 2^ℓ cells, rather than to that of the collision pattern of 2^n balls thrown at random to 2^n cells. Let $p_i(m)$ denote the expected fraction of cells that contain i balls, when we throw at random m balls into m cells. Then, $p_0(m)\approx 1/e$, for all sufficiently large m, whereas

$$
p_i(m)\approx\frac{e^{-1}}{i!}\cdot\prod_{j=1}^{i}\left(1-\frac{j-2}{m-1}\right) \qquad (5)
$$

We focus on $i\le n$ (because for $i>n$ both $p_i(2^\ell)$ and $p_i(2^n)$ are smaller than 2^{-2n}). We may ignore the (negligible in n) dependence of $p_i(2^n)$ on 2^n, but not the (noticeable) dependence of $p_i(2^\ell)$ on $2^\ell=\mathrm{poly}(n)$. Specifically, we have:

i	$p_i(2^n)$ $\approx e^{-1}/(i!)$	$p_i(n^c+1)\approx(\prod_{j=1}^i(1-(j-2)n^{-c}))\cdot p_i(2^n)$ $\approx(\prod_{j=1}^i(1-(j-2)n^{-c}))\cdot(e^{-1}/(i!))$
1	e^{-1}	$(1+n^{-c})\cdot e^{-1}$
2	$e^{-1}/2$	$(1+n^{-c})\cdot e^{-1}/2$
3	$e^{-1}/6$	$\approx(1-n^{-2c})\cdot e^{-1}/6$
4	$e^{-1}/24$	$\approx(1-1.5n^{-c})\cdot e^{-1}/24$
$i\ge 4$	$e^{-1}/(i!)$	$(1-\Theta(i^2n^{-c}))\cdot e^{-1}/(i!)$

Thus, the singleton and two-element sets are slightly over-represented in our implementation (when compared to the specification), whereas the larger sets are under-represented. In all cases, the deviation is by a factor related to $1\pm(1/\mathrm{poly}(n))$, which cannot be tolerated in a close-implementation. Thus, all that is required is to modify the function g such that it is slightly more probable to form larger collisions (inside the sets S_α's). We stress that we can easily compute all the relevant quantities (i.e., all $p_i(2^n)$'s and $p_i(2^\ell)$'s, for $i=1,...,n$), and so obtaining a close-implementation is merely a question of details, which are shortly outlined next.

Let us just sketch one possible approach. For $N\stackrel{\rm def}{=}2^n$ and $t\stackrel{\rm def}{=}2^\ell$, we have N/t sets S_α's that are each partitioned at random by the g_α's to subsets (which correspond to the sets of $\alpha\beta$'s that are mapped to the same image under g_α). Now, for a random collection of g_α's, the number of i-subsets divided by N is $p_i \stackrel{\rm def}{=} p_i(t)$ rather than $q_i \stackrel{\rm def}{=} p_i(N)$ as desired. Recall that $|p_i - q_i| \leq p_i/(t-1)$ for all $i \geq 1$, and note that $\sum_i p_i i = 1 = \sum_i q_i i$. Indeed, it is instructive to consider the fractional mass of elements that resides in i-subsets; that is, let $p'_i = p_i i$ and $q'_i = q_i i$. We need to move a fractional mass of about $1/(t-1)e$ elements from singleton subsets (resp., two-element subsets) to the larger subsets. With overwhelmingly high probability, each S_α contains more than n singleton subsets (resp., $n/2$ two-element subsets). We are going to use only these subsets towards the correction of the distribution of mass; this is more than enough, because we need to relocate only a fractional mass of $1/(t-1)e$ from each type of subsets (i.e., less than one element per a set S_α, which in turn has cardinality t). In particular, we move a fractional mass of $p'_1 - q'_1 = p'_2 - q'_2$ from singleton (resp., two-element) subsets into larger subsets. Specifically, for each $i \geq 3$, we move a fractional mass of $(q'_i - p'_i)/2$ elements residing in singletons and $(q'_i - p'_i)/2$ elements residing in two-element subsets into i-subsets.[41] This (equal contribution condition) will automatically guarantee that the mass in the remaining singleton and two-element subsets is as desired. We stress that there is no need to make the "mass distribution correction process" be "nicely distributed" among the various sets S_α's, because its affect is anyhow hidden by the application of the random permutation π_2. The only thing we need is to perform this correction procedure efficiently (i.e., for every α we should efficiently decide how to modify g_α), and this is indeed doable. ■

10 Conclusions and Open Problems

The questions that underlie our work refer to the existence of good implementations of various specifications of random objects. At the very least, we require the implementations to be computationally-indistinguishable from the corresponding specifications.[42] That is, we are interested in pseudo-implementations. Our ultimate goal is to obtain such implementations via ordinary (probabilistic polynomial-time) machines, and so we ask:

Q1: Which specifications have truthful pseudo-implementations (by ordinary machines)?

Q2: Which specifications have almost-truthful pseudo-implementations (by ordinary machines)?

Q3: Which specifications have pseudo-implementations at all (again, by ordinary machines)?

In view of Theorem 2.9, as far as Questions Q1 and Q3 are concerned, we may as well consider implementations by oracle machines (having access to a random oracle). Indeed, the key observation that started us going was that the following questions are the "right" ones to ask:

Q1r (Q1 revised)**:** Which specifications have truthful close-implementations by oracle machines (having access to a random oracle)?

Q3r (Q3 revised)**:** Which specifications have close-implementations by such oracle machines?

We remark that even in the case of Question Q2, it may make sense to study first the existence of implementations by oracle machines, bearing in mind that the latter cannot provide a conclusive positive answer (as shown in Theorem 2.11).

In this work, we have initiated a comprehensive study of the above questions. In particular, we provided a fair number of non-trivial implementations of various specifications relating to the domains of random functions, random graphs and random codes. The challenge of characterizing the class of specifications that have good implementations (e.g., Questions Q1r and Q3r) remains wide open. A good start may be to answer such questions when restricted to interesting classes of specifications (e.g., the class of specifications of random graphs having certain type of properties).

---

[41] For example, we move mass into 3-subsets by either merging three singletons or merging a singleton and a two-subset into a corresponding 3-subset, where we do three merges of the latter type per each merge of the former type. Similarly, for each $i \geq 4$, we move mass into i-subsets by merging either i singletons or $i/2$ two-subsets, while doing an equal number of merges of each type. Finally, for every $j \geq 1$, we move mass into $(2j+3)$-subsets by merging additionally created $2j$-subsets and 3-subsets (where additional 2-subsets are created by either using a 2-subset or merging two singletons, in equal proportions).

[42] Without such a qualification, the question of implementation is either meaningless (i.e., every specification has a "bad" implementation) or misses the point of generating random objects.

Limited-independence implementations. Our definition of pseudo-implementation is based on the notion of computational indistinguishability (cf. [24, 40, 20]) as a definition of similarity among objects. A different notion of similarity underlies the construction of sample spaces having limited-independence properties (see, e.g., [2, 11]). For example, we say that an implementation is k-wise close to a given specification if the distribution of the answers to any k fixed queries to the implementation is statistically close to the distribution of these answers in the specification. The study of Question Q1r is also relevant to the construction of truthful k-wise close implementations, for any $k = \text{poly}(n)$. In particular, one can show that *any specification that has a truthful close-implementation by an oracle machine, has a truthful k-wise close implementation by an ordinary probabilistic polynomial-time machine.*[43] A concrete example appears at the end of Section 5. Further study of this direction is reported in Section 11.

11 Subsequent Work

In this section, we briefly review some subsequent work that is closely related to the study initiated here.

Limited-independence implementations of random graphs. In continuation of the last paragraph of Section 10, we first mention that Alon and Nussboim [3] proved that a host of graph properties that are satisfied by truly random graphs are also satisfied by $\text{poly}(n)$*-independent random graphs* (i.e., the amount of independence is polylogarithmic in the size of the graph). These graph properties include all properties considered in our work and actually extend beyond them (e.g., optimal connectivity and jumbledness[44], which is the traditional graph theoretic notion of resemblance to a random graph). The results of [3] imply almost-truthful $\text{poly}(n)$-close implementations (by ordinary machines) of a specification that answers adjacency queries according to a uniformly distributed graph that satisfies the corresponding graph properties. Furthermore, assuming the existence of one-way functions, they obtain almost-truthful pseudo-implementations (by ordinary machines) of random graph satisfying these properties, by XORing the foregoing $\text{poly}(n)$-close implementation with a pseudo-implementation (which may not be truthful even in a weak sense).

We mention that the work of Alon and Nussboim [3] actually focuses on the general random graph model $\mathcal{G}(N,p)$, where each edge in an N-vertex graph is taken with probability $p = p(N)$ independently of all other choices (cf., the model of Erdös-Rényi as in, e.g., [8]). For any value of p (such that $p(N) = \Omega((\log N)/N)$), they show that $\text{poly}(n)$-wise independent distributions preserve the most studied graph properties of the random graph model $\mathcal{G}(N,p)$. This can be viewed as an almost-truthful "implementation of the random graph model $\mathcal{G}(N,p)$" for any p, where the specification refers to answering adjacency queries. Indeed, for small values of p (i.e., $p(N) \ll 1/\text{poly}(\log N)$), a close-implementation may return zero on all queries, but such an implementation will not satisfy the various graph properties studied in [3]. Thus, the notion of an almost-truthful implementation distinguishes the implementations provided in [3] from trivial implementations, whereas the standard notions of indistinguishability (as in Definition 2.7) fails to do so.

Implementations of the random graph model were also studied by Naor *et al.* [37], who classify graph properties according to the quantifier depth of the formulas that specify each property. Preserving high-depth properties of random graphs is introduced as an alternative measure of the quality of the implementation. Naor *et al.* [37] provide tight positive and negative results regarding the maximal D such that for every depth-D property Π it is possible to closely-implement a uniformly distributed graph having property Π. These results are incomparable with ours, since they deal with different graph properties.

Implementations of sparse random graphs. Our study of the implementation of random bounded-degree graphs was extended by Naor and Nussboim [36] who considered random graphs of polylogarithmic degree that support neighborhood queries. Clearly, efficiently answering neighborhood queries (which are answered by the list of all neighbors of the given vertex) mandates a degree bound of $\text{poly}(n) = \text{poly}(\log N)$.

[43] The claim follows by combining an implementation (by an oracle machine) that makes at most t queries to its random oracle with a sample space of $k \cdot t$-wise independent functions.

[44] Loosely speaking, jumbledness means that all vertex-sets U contain $\frac{1}{2} \cdot \binom{|U|}{2} \pm \Theta(\sqrt{N}|U|)$ edges.

Actually, the study in [36] refers to the aforementioned Erdös-Rényi model of random graphs $\mathcal{G}(N,p)$ for sufficiently low $p = p(N)$; that is, $p(N) \leq \text{poly}(\log N)/N$.

Stateful Implementations. Addressing Open Problem 5.4, Bogdanov and Wee [7] introduced the notion of stateful implementation, which (in contrast to Definition 2.6)[45] allows the implementing machine to maintain a state. That is, such a relaxed notion of an implementation may construct the object on-the-fly in respond to the queries posed (and while keeping track of all previous queries and answers). Bogdanov and Wee presented a *stateful* implementation of the "sub-cube specification" described in Open Problem 5.4, but the problem of providing a *stateless* implementation remains open.

Acknowledgments

The first two authors wish to thank Silvio Micali for discussions that took place more than two decades ago. The main part of Theorem 2.9 was essentially observed in these discussions. These discussions reached a dead-end because the notion of a specification was missing; thus, it was not understood that the interesting question is which specifications can be implemented at all (i.e., even by an oracle machine having access to a random function).

We are grateful to Noga Alon for very helpful discussions regarding random graphs and explicit constructions of bounded-degree graphs of logarithmic girth. We also thank Avi Wigderson for a helpful discussion regarding the proof of Lemma 6.3. Finally, thanks to Moni Naor for calling our attention to [14], and to Omer Reingold and S. Muthu for calling our attention to [16, Lem. 2].

We are grateful to one of the reviewers of this paper for an unusually kind, supportive, and useful report.

[45]Recall that Definition 2.6 requires that each instance of the implementation be fully determined by the machine M and the random-tape r. Similarly, Definition 2.5 requires that each instance of the implementation be fully determined by the machine M and the random oracle f.

References

[1] M. Abadi, E. Allender, A. Broder, J. Feigenbaum, and L. Hemachandra. On Generating Hard, Solved Instances of Computational Problem. In *Crypto88*, pages 297–310.

[2] N. Alon, L. Babai and A. Itai. A fast and Simple Randomized Algorithm for the Maximal Independent Set Problem. *J. of Algorithms*, Vol. 7, pages 567–583, 1986.

[3] N. Alon and A. Nussboim. k-Wise Independent Random Graphs. In *49th FOCS*, pages 813–822, 2008.

[4] E. Bach. *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms.* ACM Distinguished Dissertation (1984), MIT Press, Cambridge MA, 1985.

[5] S. Ben-David, B. Chor, O. Goldreich, and M. Luby. On the Theory of Average Case Complexity. *JCSS*, Vol. 44, No. 2, 1992, pages 193–219. Preliminary version in *21st STOC*, 1989.

[6] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SICOMP*, Vol. 13, pages 850–864, 1984. Preliminary version in *23rd FOCS*, 1982.

[7] A. Bogdanov and H. Wee. A Stateful Implementation of a Random Function Supporting Parity Queries over Hypercubes. In the proceedings of *RANDOM'04*, Springer LNCS 3122, pages 298–309, 2004.

[8] B. Bollobas. *Random Graphs.* Academic Press, 1985.

[9] B. Bollobás and P. Erdös. *Cliques in Random Graphs.* Cambridge Philosophical Society Mathematical Proc., vol. 80, 419–427, 1976.

[10] E.R. Canfield, P. Erdos, and C. Pomerance. On a Problem of Oppenheim Concerning "Factorisatio Numerorum". *Jour. of Number Theory*, Vol. 17, pages 1–28, 1983.

[11] B. Chor and O. Goldreich. On the Power of Two–Point Based Sampling. *Jour. of Complexity*, Vol 5, 1989, pages 96–106. Preliminary version dates 1985.

[12] I. Damgard. Collision Free Hash Functions and Public Key Signature Schemes. In *EuroCrypt'87*, Springer-Verlag, LNCS 304, pages 203–216.

[13] P. Erdos and H. Sachs. Reguläre Graphen gegenebener Taillenweite mit minimaler Knotenzahl. *Wiss. Z. Univ. Halle–Wittenberg, Math. Nat. R.*, 12, pages 251–258, 1963.

[14] J. Feigenbaum, S. Kannan, M. Strauss, M. Viswanathan. An Approximate L1-Difference Algorithm for Massive Data Streams. Proceedings of *40th FOCS*, pages 501–511, 1999.

[15] P. Flajolet and A.M. Odlyzko. Random mapping statistics. In *EuroCrypt'89*, Springer-Verlag, LNCS 434, pages 329–354.

[16] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, Small-Space Algorithms for Approximate Histogram Maintenance. In the proceedings of *34th STOC*, pages 389–398, 2002.

[17] O. Goldreich. A Note on Computational Indistinguishability. *IPL*, Vol. 34, pages 277–281, May 1990.

[18] O. Goldreich. *Foundation of Cryptography – Basic Tools.* Cambridge University Press, 2001.

[19] O. Goldreich. *Foundation of Cryptography – Basic Applications.* Cambridge University Press, 2004.

[20] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *JACM*, Vol. 33, No. 4, pages 792–807, 1986.

[21] O. Goldreich, and H. Krawczyk, On Sparse Pseudorandom Ensembles. *Random Structures and Algorithms*, Vol. 3, No. 2, (1992), pages 163–174.

[22] O. Goldreich and D. Ron. Property Testing in Bounded Degree Graphs. *Algorithmica*, 32 (2), pages 302–343, 2002.

[23] O. Goldreich and L. Trevisan. Three Theorems regarding Testing Graph Properties. Proceedings of *42nd FOCS*, pages 460–469, 2001. Full version in *ECCC*, TR01-010, 2001.

[24] S. Goldwasser and S. Micali. Probabilistic Encryption. *JCSS*, Vol. 28, No. 2, pages 270–299, 1984. Preliminary version in *14th STOC*, 1982.

[25] J. Hastad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SICOMP*, Volume 28, Number 4, pages 1364–1396, 1999. Preliminary versions by Impagliazzo et. al. in *21st STOC* (1989) and Hastad in *22nd STOC* (1990).

[26] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *29th STOC*, pages 220–229, 1997.

[27] W. Imrich. Explicit Construction of Regular Graphs with no Small Cycles. *Combinatorica*, Vol. 4, pages 53–59, 1984.

[28] S. Janson. The numbers of spanning trees, Hamilton cycles and perfect matchings in a random graph. *Combin. Prob. Comput.*, Vol. 3, pages 97–126, 1994.

[29] D.E. Knuth. *The Art of Computer Programming*, Vol. 2 (*Seminumerical Algorithms*). Addison-Wesley Publishing Company, Inc., 1969 (first edition) and 1981 (second edition).

[30] F. Lazebnik and V.A. Ustimenko. Explicit Construction of Graphs with arbitrary large Girth and of Large Size.

[31] L.A. Levin. Average Case Complete Problems. *SICOMP*, Vol. 15, pages 285–286, 1986.

[32] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer Verlag, August 1993.

[33] A. Lubotzky, R. Phillips, P. Sarnak, Ramanujan Graphs. *Combinatorica*, Vol. 8, pages 261–277, 1988.

[34] M. Luby and C. Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SICOMP*, Vol. 17, 1988, pages 373–386.

[35] G.A. Margulis. Explicit Construction of Graphs without Short Cycles and Low Density Codes. *Combinatorica*, Vol. 2, pages 71–78, 1982.

[36] M. Naor and A. Nussboim. Implementing Huge Sparse Random Graphs. In the *11th RANDOM*, Springer, LNCS 4627, pages 596–608, 2007.

[37] M. Naor, A. Nussboim, and E. Tromer. Efficiently Constructible Huge Graphs that Preserve First Order Properties of Random Graphs. In the *2nd TCC*, Springer, LNCS 3378, pages 66–85, 2005.

[38] M. Naor and O. Reingold. Constructing Pseudo-Random Permutations with a Prescribed Structure, *Jour. of Crypto.*, Vol. 15 (2), 2002, pages 97–102.

[39] B. Tsaban. Permutation graphs, fast forward permutations, and sampling the cycle structure of a permutation. *Journal of Algorithms*, Vol. 47 (2), pages 104–121, 2003.

[40] A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd FOCS*, pages 80–91, 1982.

Appendix A: Implementing various probability distributions

Our proof of Theorem 5.2 relies on efficient procedures for generating elements from a finite set according to two probability distributions. In both cases, we need procedures that work in time that is poly-logarithmic (rather than polynomial) in the size of the set (and the reciprocal of the desired approximation parameter). In both cases, we have close expressions (which can be evaluated in poly-logarithmic time) for the probability mass that is to be assigned to each element. Thus, in both cases, it is easy to generate the desired distribution in time that is almost-linear in the size of the set. Our focus is on generating good approximations of these distributions in time that is poly-logarithmic in the size of the set.

Indeed, the problem considered in this appendix is a special case of our general framework. We are given a specification of a distribution (i.e., each query should be answered by a sample drawn independently from that distribution), and we wish to closely-implement it (i.e., answer each query by a sample drawn independently from approximately that distribution).

A.1 Sampling the binomial distribution

We first consider the generation of elements according to the binomial distribution. For any N, we need to output any value $v \in \{0,1,...,N\}$ with probability $\binom{N}{v} \cdot 2^{-N}$. An efficient procedure for this purpose is described in Knuth [29, Sec. 3.4.1]. In fact, Knuth describes a more general procedure that, for every p, outputs the value $v \in \{0,1,...,N\}$ with probability $b_{N,p}(v) \stackrel{\text{def}}{=} \binom{N}{v} \cdot p^v(1-p)^{N-v}$. However, his description is in terms of operations with reals, and so we need to adapt it to the standard (bit-operation) model. Knuth's description proceeds in two steps:

1. In Section 3.4.1.F, it is shown how to reduce the generation of the binomial distribution $b_{N,p}$ to the generation of some *beta distributions*, which are continuous distributions over $[0,1]$ that depends on two parameters a and b.[46] The reduction involves taking $\log_2 N$ samples from certain beta distributions, where the parameters of these distributions are easily determined as a function of N. The samples of the beta distributions are processed in a simple manner involving only comparisons and basic arithmetic operations (subtraction and division).

2. In Section 3.4.1.E, it is shown how to generate any beta distribution. The generator takes a constant number of samples from the continuous uniform distribution over $[0,1]$, and produces the desired sample with constant probability (otherwise, the process is repeated). The samples of the uniform distributions are processed in a simple manner involving only comparisons and various arithmetic and trigonometric operations (including computing functions as log and tan).

The above is described in terms of real arithmetic and sampling uniformly in $[0,1]$, and provides a perfect implementation. The question is what happens when we replace the samples with ones taken from the set $\{\epsilon, 2\epsilon, ..., \lfloor 1/\epsilon \rfloor \cdot \epsilon\}$, and replace the real arithmetics with approximations up to a factor of $1 \pm \epsilon$.

[46] A beta distribution with (natural) parameters a and b is defined in terms of the accumulative distribution function

$$F_{a,b}(r) \stackrel{\text{def}}{=} a \cdot \binom{a+b-1}{a} \cdot \int_0^r x^{a-1}(1-x)^{b-1}\, dx$$

and the uniform continuous distribution is a special case (i.e., $a = b = 1$). In general, $F_{a,b}(r)$ equals the probability that the b^{th} largest of $a+b-1$ independent uniformly chosen samples in $[0,1]$ has value at most r.

Let us first consider the effect of replacing the uniform continuous distribution $U(r) = r$ by the continuous step-distribution $S_\epsilon(r) \stackrel{\text{def}}{=} \lfloor r/\epsilon \rfloor \cdot \epsilon$, where we may assume that $1/\epsilon$ is an integer. Since the variation distance between U and S_ϵ is $O(\epsilon)$, the same holds for any function applied to a constant number of samples taken from these distribution. Thus, the implementation of the beta distributions via the step-distribution S_ϵ will deviate by only $O(\epsilon)$, and using the latter to generate the binomial distribution $b_{N,p}$ only yields a deviation of $O(\epsilon \log N)$. Finally, using the *average* numerical stability of all functions employed[47] we conclude that an implementation by $O(\log(1/\epsilon))$ bits of precision will only introduce a deviation of ϵ.

A.2 Sampling from the two-set total-sum distribution

We now turn to the generation of pairs (l, r) such that $l + r = T$ and $0 \le l, r \le S$, where $T \le 2S$. Specifically, we need to produce such a pair with probability proportional to $\binom{S}{l} \cdot \binom{S}{r}$ (i.e., the number of ways to select l elements from one set of size S and r elements from another such set). (In the proof of Theorem 5.2, $S = M/2$.) Without loss of generality, we may assume that $T \le S$ (or else we select the "complementary" elements). Thus, we need to sample $r \in \{0, ..., T\}$ with probability

$$p_r = \frac{\binom{S}{T-r} \cdot \binom{S}{r}}{\binom{2S}{T}} \tag{6}$$

We wish to produce a sample with deviation at most ϵ from the correct distribution and are allowed time poly(k), where $k \stackrel{\text{def}}{=} \log(S/\epsilon)$. In case $T \le k$, we perform this task in the straightforward manner; that it, compute all the $T + 1$ probabilities p_r, and select r accordingly. Otherwise (i.e., $T > k$), we rely on the fact that p_r is upper-bounded by twice the binomial distribution of T tries (i.e., $q_r = \binom{T}{r}/2^T$). This leads to the following sampling process:

1. Select r according to the binomial distribution of T tries.
2. Compute p_r and q_r. Output r with probability $p_r/2q_r$, and go to Step 1 otherwise.

We will show (see Fact A.1 below) that $p_r \le 2q_r$ always holds. Thus, in each iteration, we output r with probability that is proportional to p_r; that is, we output r with probability $q_r \cdot (p_r/2q_r) = p_r/2$. It follows that each iteration of the above procedure produces an output with probability $1/2$, and by truncating the procedure after k iterations (and producing arbitrary output in such a case) the output distribution is statistically close to the desired one.

Fact A.1 *Suppose that $T \le S$ and $T > k$. For p_r's and q_r's as above, it holds that $p_r < 2q_r$.*

Proof: The cases $r = T$ and $r = 0$ are readily verified (by noting that $p_r = \binom{S}{T}/\binom{2S}{T} < 2^{-T}$ and $q_r = 2^{-T}$). For $r \in \{1, ..., T-1\}$, letting $\alpha \stackrel{\text{def}}{=} (S - r)/(2S - T) \in (0, 1)$, we have

$$\frac{p_r}{q_r} = \frac{\binom{S}{r} \cdot \binom{S}{T-r} / \binom{2S}{T}}{\binom{T}{r}/2^T} = 2^T \cdot \frac{\binom{2S-T}{S-r}}{\binom{2S}{S}}$$

[47] Each of these functions (i.e., rational expressions, log and tan) has a few points of instability, but we apply these functions on arguments taken from either the uniform distribution or the result of prior functions on that distribution. In particular, except for what happens in an ϵ-neighborhood of some problematic points, all functions can be well-approximated when their argument is given with $O(\log(1/\epsilon))$ bits of precision. Furthermore, the functions log and tan are only evaluated at the uniform distribution (or simple functions of it), and the rational expressions are evaluated on some intermediate beta distributions. Thus, in all cases, the problematic neighborhoods are only assigned small probability mass (e.g., ϵ in the former case and $O(\sqrt{\epsilon})$ in the latter).

$$
\begin{aligned}
&= 2^T \cdot (1+o(1)) \cdot \frac{(2\pi\alpha(1-\alpha)\cdot(2S-T))^{-1/2}\cdot 2^{\mathbf{H}_2(\alpha)\cdot(2S-T)}}{(2\pi(1/2)^2\cdot 2S)^{-1/2}\cdot 2^{\mathbf{H}_2(1/2)\cdot 2S}} \\
&= \frac{1+o(1)}{\sqrt{2\alpha(1-\alpha)\cdot\beta}} \cdot 2^{(\mathbf{H}_2(\alpha)-1)\cdot(2S-T)}
\end{aligned}
$$

where $\beta \stackrel{\rm def}{=} (2S-T)/S \geq 1$ and $\mathbf{H}_2$ is the binary entropy function. For $\alpha \in [(1/3),(2/3)]$, we can upper-bound p_r/q_r by $(1+o(1))\cdot\sqrt{9/4\beta} < 2$. Otherwise (i.e., without loss of generality $\alpha < 1/3$), we get that $\mathbf{H}_2(\alpha) < 0.92$ and $\alpha^{-1}(1-\alpha)^{-1} \leq 2S-T$, where for the latter inequality we use $1 \leq r \leq S-1$. Thus, p_r/q_r is upper-bounded by $O(\sqrt{2S-T})\cdot 2^{-\Omega(2S-T)} = O(2^{-\Omega(S)+\log S})$, which vanishes to zero with k (because $S \geq T > k$).[48] ■

A.3 A general tool for sampling strange distributions

In continuation to Appendix A.2, we state a useful lemma (which was implicitly used above as well as in prior works). The lemma suggests that poly($\log N$)-time sampling from a desired probability distribution $\{p_i\}_{i=1}^N$ can be reduced to sampling from a related probability distribution $\{q_i\}_{i=1}^N$, which is hopefully poly($\log N$)-time sampleable.

Lemma A.2 *Let $\{p_i\}_{i=1}^N$ and $\{q_i\}_{i=1}^N$ be probability distributions satisfying the following conditions:*

1. *There exists a polynomial-time algorithm that given $i \in [N]$ outputs approximations of p_i and q_i up to $\pm N^{-2}$.*
2. *Generating an index i according to the distribution $\{q_i\}_{i=1}^N$ is closely-implementable (up to negligible in* $\log N$ *deviation and in* poly($\log N$)*-time).*
3. *There exist a* poly($\log N$)*-time recognizable set $S \subseteq [N]$ such that*
   (a) *$1 - \sum_{i\in S} p_i$ is negligible in* $\log N$.
   (b) *There exists a polynomial p such that for every $i \in S$ it holds that $p_i \leq p(\log N)\cdot q_i$.*

Then generating an index i according to the distribution $\{p_i\}_{i=1}^N$ is closely-implementable.

Proof: Without loss of generality, S may exclude all i's such that $p_i < N^{-2}$. For simplicity, we assume below that given i we can exactly compute p_i and q_i (rather than only approximate them within $\pm N^{-2}$). Let $t \stackrel{\rm def}{=} p(\log N)$. The sampling procedure proceeds in iterations, where in each iteration i is selected according to the distribution $\{q_i\}_{i=1}^N$, and is output with probability p_i/tq_i if $i \in S$. (Otherwise, we proceed to the next iteration.) Observe that, conditioned on producing an output, the output of each iteration is in S and equals i with probability $q_i\cdot(p_i/tq_i) = p_i/t$. Thus, each iteration produces output with probability $\sum_{i\in S} p_i/t > 1/2t$, and so halting after $O(t\log(1/\epsilon))$ iterations we produce output with probability at least $1-\epsilon$. For any $i \in S$, the output is i with probability $(1\pm\epsilon)\cdot p_i/\rho$, where $\rho \stackrel{\rm def}{=} \sum_{j\in S} p_j$. Setting ϵ to be negligible in $\log N$, the lemma follows. ■

A typical application of Lemma A.2 is to the case that for each $i \in [N]$ the value of p_i can be approximated by one out of $m = $ poly($\log N$) predetermined p_j's. Specifically:

[48] In fact, it holds that $p_r \leq \sqrt{2}\cdot q_r$ for all r's, with the extreme value obtained at $r = T/2$ (and $T = S$), where we have $\alpha = 1/2$ (and $\beta = 1$).

Corollary A.3 *Let $\{p_i\}_{i=1}^N$ be a probability distribution and $S \subseteq [N]$ be a set satisfying Conditions* (1) *and* (3a) *of Lemma A.2. Suppose that, for $m, t = \mathrm{poly}(\log N)$, there exists an efficiently constructible sequence of integers $1 = i_1 < i_2 < \cdots < i_m = N$ such that for every $j \in [m-1]$ and $i \in [i_j, i_{j+1}] \cap S$ it holds that $p_{i_j}/t < p_i < t \cdot p_{i_j}$. Then generating an index i according to the distribution $\{p_i\}_{i=1}^N$ is closely-implementable.*

Proof: For every $j \in [m-1]$ and $i \in [i_j, i_{j+1}] \cap S$, define $p'_i = p_{i_j}$ and note that $p'_i/t < p_i < t \cdot p'_i$. Let $p' = \sum_{i \in S} p'_i$, and note that $p' < t$. Now, define $q_i = p'_i/p'$ for every $i \in S$, and $q_i = 0$ otherwise. Then, for every $i \in S$, it holds that $p_i < t \cdot p'_i = t \cdot p' \cdot q_i < t^2 q_i$. Since these q_i's satisfy Conditions (1), (2) and (3b) of Lemma A.2, the corollary follows. ■

Appendix B: Implementing a Random Bipartite Graph

Following the description in Section 6, we present a close-implementation of random bipartite graphs. Two issues arise. Firstly, we have to select the proportion of the sizes of the two parts, while noticing that different proportions give rise to different number of graphs. Secondly, we note that a bipartite graph uniquely defines a 2-partition (up to switching the two parts) only if it is connected. However, since all but a negligible fraction of the bipartite graphs are connected, we may ignore the second issue, and focus on the first one. (Indeed, the rest of the discussion is slightly imprecise because the second issue is ignored.)

For $i \in [\pm N]$, the number of $2N$-vertex bipartite graphs with $N + i$ vertices on the first part is

$$\binom{2N}{N+i} \cdot 2^{(N+i)\cdot(N-i)} \leq \binom{2N}{N} \cdot 2^{N^2 - i^2}$$

where equality holds for $i = 0$ and approximately holds (i.e., up to a constant factor) for $|i| = \sqrt{N}$. Thus, all but a negligible fraction of the $2N$-vertex bipartite graphs have $N \pm \log_2 N$ vertices on each part. That is, we may focus on $O(\log N)$ values of i. Indeed, for each $i \in [\pm \log_2 N]$, we compute $T_i \stackrel{\text{def}}{=} \binom{2N}{N+i} \cdot 2^{N^2 - i^2}$, and $p_i = T_i/T$, where $T \stackrel{\text{def}}{=} \sum_{j=-\log_2 N}^{\log_2 N} T_j$. Next, we select i with probability p_i, and construct a random $2N$-vertex bipartite graph with $N + i$ vertices on the first part as follows:

- As in Section 6, we use the function f_1 to implement a permutation π. We let $S \stackrel{\text{def}}{=} \{v : \pi(v) \in [N + i]\}$, and $\chi_S(i) \stackrel{\text{def}}{=} 1$ if and only if $i \in S$.
- As in Section 6, we answer the query (u, v) by 0 if $\chi_S(u) = \chi_S(v)$ and according to the value of f_2 otherwise.

Appendix C: Various Calculations

Calculations for the proof of Lemma 6.3

The proof of Lemma 6.3 refers to the following known fact:

Fact C.1 *Let X be a random variable ranging over some domain D, and suppose that $\mathbf{H}(X) \geq \log_2 |D| - \epsilon$. Then X is at statistical distance at most $O(\sqrt{\epsilon})$ from the uniform distribution over D.*

Proof: Suppose that X is at statistical distance δ from the uniform distribution over D. Then, there exists a $S \subset D$ such that $|\mathbf{Pr}[X \in S] - (|S|/|D|)| = \delta$, and assume without loss of generality that $|S| \geq |D|/2$. Note that either for each $e \in S$ it holds that $\mathbf{Pr}[X = e] \geq 1/|D|$ or for each $e \in S$ it holds that $\mathbf{Pr}[X = e] \leq 1/|D|$. By removing the $|S| - (|D|/2)$ elements of smallest absolute difference (i.e., smallest $|\mathbf{Pr}[X = e] - (1/|D|)|$), we obtain a set S' of size $|D|/2$ such that $|\mathbf{Pr}[X \in S'] - (|S'|/|D|)| \geq \delta/2$. The entropy of X is maximized when it is uniform both on S' and on $D \setminus S'$. Thus:

$$\begin{aligned}
\mathbf{H}(X) &\leq \mathbf{H}_2(\mathbf{Pr}[X \in S']) + \mathbf{Pr}[X \in S'] \cdot \mathbf{H}(X|X \in S') + \mathbf{Pr}[X \in D \setminus S'] \cdot \mathbf{H}(X|X \in D \setminus S') \\
&= \mathbf{H}_2\left(\frac{1}{2} + \frac{\delta}{2}\right) + \log_2(|D|/2) \\
&= 1 - \Omega(\delta^2) + \log_2(|D|/2)
\end{aligned}$$

We get that $\mathbf{H}(X) \leq \log_2|D| - c \cdot \delta^2$, for some universal $c > 0$. Combining this with the hypothesis that $\mathbf{H}(X) \geq \log_2|D| - \epsilon$, we get that $\epsilon \geq c \cdot \delta^2$, and $\delta \leq \sqrt{\epsilon/c}$ follows. ■

Calculations for the proof of Theorem 6.6

In order to complete the proof of Part 2 of Theorem 6.6, we prove the following claim.

Claim C.2 *Let $c(N) = (2 - o(1))\log_2 N$ be as in Theorem 6.6, and let $T \stackrel{\text{def}}{=} \lceil N/c(N) \rceil$. Consider any fixed partition $(S^{(1)}, ..., S^{(T)})$ of $[N]$ such that $|S^{(i)}| = c(N)$, for every $i < T$, and $|S^{(T)}| \leq c(N)$. Consider a graph selected as follows:*

- *Each $S^{(i)}$ is an independent set.*
- *For $k = 2\binom{c(N)+2}{2}$, the edges between vertices residing in different $S^{(i)}$'s are determined by a k-wise independent sequence of unbiased bits.*

Then, with probability at least $1 - (N^{-\Theta(1)})$, the graph has no independent set of size $c(N) + 2$.

Applying Claim C.2 to any partition $(S_r^{(1)}, ..., S_r^{(T)})$ fixed at the end of the proof of Theorem 6.6, it follows that the graph g^{color} contains no independent set of size $c(N) + 2$. Part 2 of Theorem 6.6 follows.

Proof: We will show that the expected number E of independent sets of size $c(N) + 2$ is $N^{-\Omega(1)}$, and the claim will follow. Denoting $c \stackrel{\text{def}}{=} c(N)$ and $c' \stackrel{\text{def}}{=} c + 2$, we consider an arbitrary vertex-set V of size c' (so V is a potential independent-set). The analysis bounds the contribution of various vertex-sets V (to the entire expectation E) according to the sizes of the intersections $V \bigcap S^{(j)}$.

We shall use the following notation. For any V as above, we let $n(V)$ denote the *number* of non-empty intersections $V \bigcap S^{(j)}$, and let $s(V)$ denote the *size* of the largest intersection. Next, let A_s denote the collection of all vertex-sets V for which $s(V) = s$, and let B_n denote the collection of those vertex-sets V for which $n(V) = n$. Finally, let p_V denote the probability that V induces an independent-set, and let $P_s \stackrel{\text{def}}{=} \max_{V \in A_s}\{p_V\}$ and $Q_n \stackrel{\text{def}}{=} \max_{V \in B_n}\{p_V\}$. The following facts summarize a few useful upper-bounds.

Fact C.2.1 *For any $1 \leq s \leq c$ and any $1 \leq n \leq c'$ it holds that:*

1. $|A_s| \leq \lceil \frac{N}{c} \rceil \binom{c}{s} \binom{N}{c'-s} = N^{(2\log_2 N) - s + o(\log N)}$.

2. $|B_n| \leq \binom{\lceil \frac{N}{c} \rceil}{n}\binom{c+1}{n-1}c^{c'} = N^{n+o(\log N)}$.

Fact C.2.2 *For any $1 \leq s \leq c$ and any $3 \leq n \leq c'$ we have*

1. $P_s \leq 2^{-(c'-s)\cdot s}$.
2. $P_s \leq N^{-(c'-s)+o(\log N)}$.
3. $Q_n \leq 2^{-\binom{c+2}{2}+\binom{c-n+3}{2}}$.
4. $Q_n \leq N^{-n(2-\frac{n}{2\log_2 N})+o(\log N)}$.

(Proving Facts C.2.1 and C.2.2 is deferred to the end of this subsection.) The desired upper-bound on the expected number E of independent-sets is established via a case analysis where we separately handle the contribution of various vertex-sets V to the expectation E, according to the values of $s(V)$ and $n(V)$. For the rest of the analysis, we fix an arbitrary constant $\Delta \in (0, 1/6)$.

Case 1 – Large maximal intersection: $s \geq (\frac{3}{2}+\Delta)\log_2 N$. By the first items in Facts C.2.1 and C.2.2 we take $\mathbf{E}_s \stackrel{\text{def}}{=} [\lceil \frac{N}{c} \rceil \binom{c}{s}\binom{N}{c'-s}]2^{-s[c'-s]}$ as an upper-bound on the expected number of independent-sets that are induced by sets V with $s(V) = s$. We claim that for large values of s, $\mathbf{E}_s$ is maximized when s is maximal, namely, when $s = c$. Indeed,

$$\begin{aligned}
\frac{\mathbf{E}_{s+1}}{\mathbf{E}_s} &= \left[\frac{(c-s)}{(s+1)} \cdot \frac{(c'-s)}{(N-c+s-1)}\right] \cdot 2^{2s}2^{-(c+1)} \\
&\geq \left[\frac{N^{o(1)}}{N^{o(1)}} \cdot \frac{N^{o(1)}}{N^{1-o(1)}}\right] \cdot 2^{2(\frac{3}{2}+\Delta)\log_2 N}2^{(-2+o(1))\log_2 N} \\
&= \left[N^{-1-o(1)}\right] N^{3+2\Delta}N^{-2+o(1)} = N^{2\Delta - o(1)} \gg 1,
\end{aligned}$$

where the first inequality uses the fact that s is large. Thus for sufficiently large N the maximal term is $\mathbf{E}_c = \left[\lceil \frac{N}{c} \rceil \cdot \binom{N}{c'-c}\right] 2^{-c[c'-c]} < [N \cdot N^2]2^{-2([2-o(1)]\log_2 N)} = N^{-1+o(1)}$. Consequently, as there are only $\Theta(\log N)$ possible values of s, the expected number of independent-sets with large s is bounded by $N^{-\Theta(1)}$.

Case 2 – Large number of intersections: $n \geq (1+\Delta)\log_2 N$. Analogously to case 1, we combine the second item in Fact C.2.1 with the third item in Fact C.2.2 to deduce that $\bar{\mathbf{E}}_n \stackrel{\text{def}}{=} \left[\binom{\lceil \frac{N}{c} \rceil}{n}\binom{c+1}{n-1}c^{c'}\right]$ $\cdot 2^{-\binom{c+2}{2}+\binom{c-n+3}{2}}$ upper-bounds the expected number of independent-sets that are induced by sets V with $n(V) = n \geq 3$. We show that for large values of n, $\bar{\mathbf{E}}_n$ is maximized when n is maximal, namely, when $n = c'$. Indeed,

$$\begin{aligned}
\frac{\bar{\mathbf{E}}_{n+1}}{\bar{\mathbf{E}}_n} &= \left[\frac{\lceil \frac{N}{c} \rceil - n}{n+1} \cdot \frac{c-n+2}{n}\right] \cdot 2^n 2^{-(c+2)} \\
&\geq \left[\frac{N^{1-o(1)}}{N^{o(1)}} \cdot \frac{N^{o(1)}}{N^{o(1)}}\right] \cdot 2^{(1+\Delta)\log_2 N}2^{[-2+o(1)]\log_2 N} \\
&= \left[N^{1-o(1)}\right] \cdot N^{(1+\Delta)}N^{-2+o(1)} = N^{\Delta - o(1)} \gg 1.
\end{aligned}$$

Thus for sufficiently large N the maximal term is $\bar{\mathbf{E}}_{c'}$. To bound $\bar{\mathbf{E}}_{c'}$ we use the notation $\Psi \stackrel{\rm def}{=} \binom{N}{c'} 2^{-\binom{c+2}{2}}$ and note that

$$\begin{aligned}
\bar{\mathbf{E}}_{c'} &= \binom{\lceil \frac{N}{c} \rceil}{c'} c^{c'} 2^{-\binom{c+2}{2}} \\
&= \left[\binom{N}{c'}^{-1} \binom{\lceil \frac{N}{c} \rceil}{c'} \right] c^{c'} \cdot \binom{N}{c'} 2^{-\binom{c+2}{2}} \\
&= \left[\prod_{i=0}^{c'-1} \frac{\lceil \frac{N}{c} \rceil - i}{N - i} \right] c^{c'} \cdot \Psi \\
&= \left[\frac{[1 \pm o(1)]}{c^{c'}} \right] c^{c'} \cdot \Psi \\
&\leq [1 + o(1)] \cdot N^{-1+o(1)} = N^{-\Theta(1)},
\end{aligned}$$

where the last inequality uses the fact that $\Psi \leq N^{-1+o(1)}$ (taken, again from [9]). Thus, as there are only $\Theta(\log N)$ possible values of n, the expected number of independent-sets with large n is bounded by $N^{-\Theta(1)}$.

Case 3 – Medium number of intersections: $\Delta \log_2 N \leq n \leq (1 + \Delta) \log_2 N$. We shall actually establish the claim for $\Delta \log_2 N \leq n \leq (2 - \Delta) \log_2 N$. By the second item in Fact C.2.1 and the last item in Fact C.2.2 the expected number of independent-sets that are induced by sets V with $n(V) = n \geq 3$ is bounded by

$$\begin{aligned}
&N^{n+o(\log N)} N^{-n(2-\frac{n}{2\log_2 N})+o(\log N)} \\
&\quad \leq N^{n(-1+\frac{(2-\Delta)\log_2 N}{2\log_2 N})+o(\log N)} \\
&\quad = N^{-\frac{\Delta n}{2}+o(\log N)} = N^{-\Theta(\log N)},
\end{aligned}$$

where the first inequality employs the fact that n is medium and the final equality uses $n = \Theta(\log N)$. Therefore, as there are only $\Theta(\log N)$ possible values of n, the expected number of independent-sets with medium n is bounded by $N^{-\Theta(\log N)}$.

Case 4 – Small intersections and a small number of intersections: $n \leq \Delta \log_2 N$ and $s \leq (\frac{3}{2} + \Delta) \log_2 N$. We shall actually establish the claim for $n \leq (\frac{1}{2} - 2\Delta) \log_2 N$ (and $s \leq (\frac{3}{2} + \Delta) \log_2 N$). Fix any n and s as above and let $\mathbf{E}_{s,n}$ denote the expected number of independent-sets that are induced by vertex-sets $V \in A_s \bigcap B_n$. By the second items in Facts C.2.1 and C.2.2 we get

$$\begin{aligned}
\mathbf{E}_{s,n} &\leq N^{n+o(\log N)} N^{-[c'-s]+o(\log N)} \\
&\leq N^{(\frac{1}{2}-2\Delta)\log_2 N+o(\log N)} N^{-[2-(\frac{3}{2}+\Delta)]\log_2 N+o(\log N)} \\
&= N^{-[\Delta+o(1)]\log_2 N} = N^{-\Theta(\log N)},
\end{aligned}$$

where the second inequality uses the fact that s and n are both small. Thus, as there are only $\Theta(\log^2 N)$ possible pairs (s, n), the expected number of independent-sets with small s and small n is bounded by $N^{-\Theta(\log N)}$.

These four cases handle all possible pairs (s, n), so a $N^{-\Omega(1)}$ bound on the expected number of independent-sets is achieved, and the current claim (i.e., Claim C.2) follows once Facts C.2.1 and C.2.2 are proved.

Proving Fact C.2.1. To derive the upper bounds on $|A_s|$ we choose a vertex-set $V \in A_s$ as follows. There are $\lceil \frac{N}{c} \rceil$ possible choices for the forced independent set $S^{(j)}$ that achieves the maximal intersection with V. Then, there are at most $\binom{c}{s}$ choices for the vertices of $V \bigcap S^{(j)}$. Finally, there are less than $\binom{N}{c'-s}$ possible choices for the vertices of $V \backslash S^{(j)}$. Thus $|A_s| \leq \lceil \frac{N}{c} \rceil \binom{c}{s} \binom{N}{c'-s}$. The $|A_s| \leq N^{[2\log_2 N - s] + o(\log N)}$ bound follows from the above by observing that $\binom{c}{s} < 2^c = 2^{[2-o(1)]\log_2 N} = N^{o(\log N)}$, and that $\binom{N}{c'-s} < N^{c'-s} = N^{[2-o(1)]\log_2 N - s}$.

To prove the upper bounds on $|B_n|$ we choose a vertex-set $V \in B_n$ as follows. There are precisely $\binom{\lceil \frac{N}{c} \rceil}{n}$ possible choices for the forced independent sets $S^{(i_1)}, ..., S^{(i_n)}$ that intersect V. Once these sets $S^{(i_j)}$ are fixed, there are exactly $\binom{c+1}{n-1}$ possible choices for the cardinalities $r_1 \stackrel{\text{def}}{=} |V \bigcap S^{(i_1)}|, \cdots, r_n \stackrel{\text{def}}{=} |V \bigcap S^{(i_n)}|$. Finally, given these cardinalities, there are no more than $\prod_{i=1}^n \binom{c}{r}_i < \prod_{i=1}^n c^{r_i} = c^{c'}$ choices for the vertices themselves. This implies that $|B_n| \leq \binom{\lceil \frac{N}{c} \rceil}{n} \binom{c+1}{n-1} c^{c'}$. The $|B_n| \leq N^{n+o(\log N)}$ bound is derived by observing that $\binom{\lceil \frac{N}{c} \rceil}{n} < N^n$, and that $\binom{c+1}{n-1} c^{c'} < (c+1)^{n-1+c'} = \Theta(\log N)^{\Theta(\log N)} = N^{\Theta(\log\log N)}$.

Proving Fact C.2.2. Fix an arbitrary vertex-set $V \in A_s \bigcap B_n$ and consider the set $I(V)$ of internal random edges of V; that is, $I(V) \stackrel{\text{def}}{=} \{\{v, w\} : \exists i \neq j \text{ s.t. } v \in V \cap S^{(i)} \wedge w \in V \cap S^{(j)}\}$. By the k-wise independence of our graph, the probability that V induces an independent-set equals $2^{-|I(V)|}$. Note that even by considering only the edges that connect the largest intersection, $V \bigcap S^{(j)}$, to $V \setminus S^{(j)}$ we get $|I(V)| \geq s \cdot (c' - s)$, and Item 1 follows. For Item 2, note that since $s(V) = s$, then each of the c' vertices $v \in V$ contributes at least $(c' - s)$ edges to $I(V)$. As each edge is counted twice, we get $|I(V)| \geq \frac{1}{2} \cdot (c' - s)c'$, so $P_s \leq 2^{-\frac{1}{2} \cdot (c'-s) \cdot (2-o(1))\log_2 N}$. Item 2 follows.

For Items 3–4 we will demonstrate that for any fixed $n \geq 3$, the maximal probability Q_n is achieved by a vertex-set V where all non-empty intersections $V \bigcap S^{(j)}$ are of size 1, except the largest intersection. Indeed, assume w.l.o.g. that V has decreasing intersection's sizes $r_1 \geq \cdots \geq r_n > 0$. Now assume that $r_2 \geq 2$. Since $n \geq 3$ and $\sum_{i=1}^n r_i = c + 2$, then $r_1 + 1 \leq c$. Thus there exists another vertex-set V' with intersections of sizes $r_1 + 1, r_2 - 1, r_3, \cdots, r_n$. It's readily verified the the probability that V' induces an independent-set is at least twice the probability that V does. Therefore the maximal probability Q_n is achieved when $r_2 < 2$ so $r_2 = \cdots = r_n = 1$ and $r_1 = c + 3 - n$. Then $|I(V)| = \binom{c+2}{2} - \binom{r_1}{2} = \binom{c+2}{2} - \binom{c-n+3}{2}$ and Item 3 follows. Item 4 is derived from Item 3 since

$$\begin{aligned}
\binom{c+2}{2} - \binom{c-n+3}{2} &= n\left(c - \frac{n}{2}\right) + \frac{1}{2}(5n - 2c - 4) \\
&= n\log_2 N\left([2 - o(1)] - \frac{n}{2\log_2 N}\right) + \frac{1}{2}(5n - 2c - 4) \\
&= n\log_2 N\left(2 - \frac{n}{2\log_2 N}\right) \pm o(\log^2 N).
\end{aligned}$$

This establishes Fact C.2.2.

Having established Facts C.2.1 and C.2.2, the entire claim (i.e., Claim C.2) follows. ■

Appendix D: A strengthening of Proposition 2.15

The hypothesis of Part 2 of Proposition 2.15 requires the existence of one-way functions, or equivalently the ability to generate hard-instances (to NP-problems) *along with corresponding solutions* (cf. [18, Sec 2.1]). A seemingly weaker condition, which is in the spirit of Levin's theory of average-case complexity [31] (see also [5]), is the ability to generate hard-instances to NP-problems. Specifically:

Definition D.1 (generating hard instances): *A probabilistic polynomial-time algorithm G is called a* generator of hard instances for a set *S if for every probabilistic polynomial-time algorithm A the probability that A correctly decides whether or not $G(1^n)$ is in S is bounded away from 1. That is, there exists a polynomial p such that for all sufficiently large n's it holds that*

$$\mathbf{Pr}_{x\leftarrow G(1^n)}[A(x)=\chi_S(x)] \;<\; 1-\frac{1}{p(n)}$$

where $\chi_S(x)=1$ if $x\in S$ and $\chi_S(x)=0$ otherwise.

Definition D.1 only requires that hard instances be generated with "noticeable" probability. Note that the existence of one-way functions (even weak ones) implies the ability to generate hard instances to NP-problems. The converse is not known to hold. Thus, the following result strengthens Part 2 of Proposition 2.15.

Proposition D.2 *Assuming the existence of generators of hard instances for NP-problems, there exist specifications that cannot be pseudo-implemented.*

Proof: Let L be an NP-set that has a generator G of hard instances, let R be the corresponding witness relation (i.e., $L=\{x:\exists y \text{ s.t. } (x,y)\in R\}$), and $R(x)\stackrel{\rm def}{=}\{y:(x,y)\in R\}$. Consider the specification that answers query x with a uniformly distributed $y\in R(x)$ if $R(x)\neq\emptyset$ and with a special symbol otherwise. We will show that this specification cannot be pseudo-implemented.

Let I be an arbitrary implementation of the above specification, and consider a distinguisher that, for parameter n, makes the query $x\leftarrow G(1^n)$, obtains the answer y, and outputs 1 if and only if $(x,y)\in R$ (which is polynomial-time decidable). When this distinguisher queries the specification, it outputs 1 with probability that equals $\rho\stackrel{\rm def}{=}\mathbf{Pr}[G(1^n)\in L]$. Assume, towards the contradiction, that when the distinguisher queries I it outputs 1 with probability that at least $\rho-\mu(n)$, where μ is a negligible function. In such a case we obtain a probabilistic polynomial-time algorithm that violates the hypothesis that G generates hard instances: Specifically, consider an algorithm A such that $A(x)$ answers 1 if and only if $(x,I(x))\in R$, and note that A is always correct when it outputs 1. Thus,

$$\begin{aligned}\mathbf{Pr}_{x\leftarrow G(1^n)}[A(x)=\chi_L(x)] &= \mathbf{Pr}[x\in L \,\wedge\, A(x)=1]+\mathbf{Pr}[x\notin L]\cdot\mathbf{Pr}[A(x)=0|x\notin L]\\ &= \mathbf{Pr}[x\in L \,\wedge\, (x,I(x))\in R]+(1-\rho)\cdot\mathbf{Pr}[(x,I(x))\notin R|x\notin L]\\ &\geq (\rho-\mu(n))+(1-\rho)\cdot 1 \;=\; 1-\mu(n)\end{aligned}$$

(which violates the hypothesis). We conclude that the implementation I cannot be computationally indistinguishable from the specification, and the proposition follows. ■