Available online at www.sciencedirect.com

SCIENCE DIRECT®

Information and Computation 189 (2004) 135–159

Information and Computation

www.elsevier.com/locate/ic

Fast approximate probabilistically checkable proofs☆

Funda Ergün,[a] Ravi Kumar,[b,*] and Ronitt Rubinfeld[c]

[a] *Case Western Reserve University, Cleveland, OH 44106, USA*
[b] *IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA*
[c] *NEC Laboratories America, 4 Independence Way, Princeton, NJ 08540, USA*

Received 17 September 2003

Abstract

We investigate the question of when a verifier, with the aid of a proof, can reliably compute a function *faster* than it can without the proof. The proof system model that we use is based on a variant of the Probabilistically Checkable Proofs (PCP) model, in which a verifier can ascertain the correctness of the proof by looking at very few locations in the proof. However, known results in the PCP model require that the verifier spend time linear in the size of the input in order to determine where to query the proof. In this work, we focus on the case when it is enough for the verifier to know that the answer is *close* to correct, and develop an approximate PCP model. We construct approximate PCPs for several optimization problems, in which the total running time of the verifier is significantly less than the size of the input. For example, we give polylogarithmic time approximate PCPs for showing the existence of a large cut, or a large matching in a graph, and a small bin packing. In the process, we develop a set of tools for use in constructing these proof systems.
© 2003 Elsevier Inc. All rights reserved.

AMS classification: 68Q25; 68Q10; 68R01

Keywords: Property testing; Proof-assisted property testing; Probabilistically checkable proofs; Sub-linear time algorithms; Optimization problems

☆This work was partially supported by ONR N00014-97-1-0505, MURI, NSF Career Grant CCR-9624552, and an Alfred P. Sloan Research Award. A preliminary abstract describing results in this work has appeared in the *Proceedings of 31st Annual ACM Symposium on Theory of Computing*, 1999, pp. 41–50.

*Corresponding author. Fax: 1-408-927-3215.

E-mail addresses: afe@eecs.cwru.edu (F. Ergün), ravi@almaden.ibm.com (R. Kumar), ronitt@nec-labs.com (R. Rubinfeld).

0890-5401/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2003.09.005

1. Introduction

Consider the following scenario: A client sends a computational request to a "consulting" company on the internet, by specifying an input and a computational problem to be solved. The company then computes the answer and sends it back to the client. This scenario is of interest whenever the company can help a client reliably find the answer to a function faster than the client could compute the function on its own, or whenever the client does not possess the code required to solve the computational problem. An obvious issue that arises, especially in the case that the company does not have a well-established reputation, is: why should the client believe the answer to be correct? Surprising results on Probabilistically Checkable Proof (PCP) systems show that there is a format in which the company can write a proof of correctness of the result such that the proof can be verified by a client (verifier) that looks at only a constant number of bits of the proof (for example, see [1,36]). However, in these approximate PCPs, the running time of the verifier has at least linear dependence on the size of the theorem statement, which in turn is necessarily at least as large as the input data.

In this paper we study the setting in which the computations are performed on large data sets. In this setting, it is desirable to find proof systems for clients that do very little work, running in time *sublinear* in the size of the data set. While this may at first seem to be an impossible task, we show that when it is enough for the client to know that the answer is *close* to correct, then in many cases it is possible to write the proof in a format where the verifier requires sublinear, in some cases even constant or polylogarithmic, time to verify the proof. To illustrate our notion of close, consider a graph that has a cut of size at k, the client may be willing to accept a proof that only ascertains that the size of the cut is at least $(1-\epsilon)k$.

Our results. The model we consider in this paper, described in Section 2, is based on the model of PCP [19] with modifications borrowed from the models of CS proofs [34], program checking [5], approximate program checking [22], property testing [23,37], and spot-checking [14]. We concentrate on minimizing the running time of the verifier. All of the running time bounds in our results yield corresponding upper bounds on the query complexity that are no better and often somewhat worse than those that would be attained by using the known PCP results. Furthermore, because our upper bounds apply only to promise problems, where the behavior of the verifier is guaranteed only for inputs that are either in the language or very far from being in the language, our results do not have any implications to the complexity of proof systems for problems studied in the traditional PCP literature (for example, see [1]).

In the PCP model, there are protocols that work for sets of problems defined by complexity classes such as NP, NEXP. In this model, we do not know of any such general purpose protocols. In fact, as we will see, there are some very simple and efficient protocols for approximations of NP-complete problems, whereas there are other polynomial-time problems for which we know of no protocol with a sublinear time verifier. This is similar to the situation in the area of property testing [23,37], where there are constant time property testers for several NP-complete problems [23], but for several other "easy" problems it is known that property testers require time that has some dependence on the input size (cf. [16]).

We begin by considering problems that return approximations of optimal solutions for combinatorial optimization problems. We give efficient proof systems for proving good lower bounds on the solution quality to constraint satisfaction problems, including Max Cut and

Max SAT, to a polylogarithmic time verifier. We next show how to prove the existence of a near optimal solution of a sparse fractional packing problem to a polylogarithmic time verifier. The techniques behind our approximate PCP for fractional packing can be used for several other problems. For example, it is possible to prove the existence of a large flow, a large matching, or a small bin packing in such a way that the verifier need only spend time nearly linear in the number of vertices (which is sublinear for graphs that are not sparse) in the first case and polylogarithmic time in the latter cases. The size of the proof is nearly linear in the size of the solution to the corresponding search problem and the proof can be computed efficiently by the prover. In all of the above approximate PCPs it is also possible to prove the existence of sub-optimal solutions, i.e., if there exists a solution of value v, then there is a proof that convinces the verifier of the existence of a solution of value at least $(1-\epsilon)v$.

We also consider a different type of approximation problem within this model, in particular the task of property testing. That is, given an object, we would like to know if the object is close to having the relevant property, i.e., whether it is close with respect to some notion of distance to some other object that has the property. We give examples of properties for which there is a proof system for which the verifier is provably more efficient than any property tester for the same property.

We develop a new set of tools for use in constructing these proof systems. For example, we give an approximate PCP for estimating lower bounds on sums of n inputs where the verifier runs in constant time. We develop a constraint enforcement protocol that allows the verifier to ensure that linear upper bound constraints are satisfied without looking at all of the variables involved.

Some possible applications. Let us mention two examples of properties of massive data sets to which our proof systems apply.

(1) *Quality of service in networks.* A company wants to convince a client that the company's network is capable of handling a large sample load provided by the client. The above techniques could be used to convince the client that at least $1-\epsilon$ fraction of the load can be routed, such that the running time of the client is $O(d(\log n)/\epsilon)$ where d is the diameter of the network (typically much smaller than the number of nodes in the network).
(2) *Website hits.* In order to prove the popularity of their website to advertisers, a company may present a list of machines that have accessed their website. The list may be made longer by either adding fake entries (machines that did not access the website or do not exist) or by duplicating the existing legal entries. Assuming that the advertisers have a way of detecting fake entries, standard sampling methods can be used to ensure that at most an $\epsilon/2$ fraction of the entries are fake in $O(1/\epsilon)$ time. Methods given in Section 3.2 allow the advertisers to ensure that at most an $\epsilon/2$ fraction of the entries are duplicates in $O(1/\epsilon)$ time.

Using PCP over a communication channel. When a verifier reads a proof over a communication channel (such as the internet), it may not be appropriate for the verifier to assume that the proof does not change during the interaction. However, for many of the PCP protocols, including most of the protocols in this paper, it is easy to see that the ability to change the proof during the interaction allows one to convince the verifier of an incorrect proof. Furthermore, it may be infeasible for the verifier to download the whole proof Π before beginning the checking phase. Instead, the verifier may want some assurance that the bits of the proof Π do not change depending on the past communication (as in the oracle prover model). One possibility is to use a trusted third party: Π is transmitted to the third party, and the verifier interacts with the third party assuming that it has no reason to change bits of the proof. Alternatively, if one assumes a bound on the running time of the entity that has produced the proof Π, then it is possible to force the entity to commit to the proof in such a way that only entities that are computationally more powerful than the allowed bound are able to change the proof in a convincing way. One can use the commitment methods of [33] in this setting, as was described in [30,31,34].

Related work. Probabilistically checkable proof systems [19] can be used to convince a polynomial-time verifier of the correctness of a decision problem computation. The PCP model is equivalent in power to multiple prover proof systems [4] (see also [2,15]) and to the oracle prover model [19]. It is known that a proof of membership in any NP language can be written such that only a constant number of locations of the proof need to be seen in order to verify the correctness of the proof [1]. Thus we have a good understanding of the set of problems for which it is feasible to find proof systems in which the verifier is efficient and the number of times that the verifier can query the proof is limited. Note that the protocols in the aforementioned results all require that the verifier look at the whole input in order to choose the locations in the proof to query, and thus do not give sublinear time protocols.

However, if the input were presented in a good error-correcting code format, there is a verifier for such proofs whose running time is independent of the size of the input [3]. This can be considered the first (and to our knowledge the only other) result on probabilistically checkable proofs with sublinear time verifiers. Our work differs in that it focuses on inputs that are presented in relatively standard formats. The result of Babai et al. [3] can be used to provide a nontrivial, though not sublinear, bound on the running time of any verifier: since inputs can be converted to such an error-correcting code format in linear time [39], it follows that it is possible to construct proof systems for any proof in a reasonable formal system with an $O(n + \log \ell)$-time verifier, where n is the length of the theorem and ℓ is the length of the proof.

Program result checking [5] and self-testing/correcting techniques [6,32] were introduced so that a client could ensure the correctness of a solution to a computation. Program result checkers yield a special type of proof system for function computations where the proof consists of a list of all possible instances of the same computational problem, i.e., the value of the desired function for each possible input. It is easy to see that all result checkers as well as result checkers in the library setting [6] satisfy the requirements of the model used here, although none have verifiers which run in sublinear time.

Proving that results are approximately correct is also related to approximate checking [22], property testing [23,37], and spot-checking [14], where the goal is to determine whether an answer is close to correct for various interesting notions of closeness. All approximate checkers satisfy the requirement of the model here, although again, none of the previously known checkers has a sublinear time verifier. Conversely, all of our results can be restated as property testers or spot-checkers that use the additional aid of a proof.

Several other works have looked at PCPs with resource limited verifiers, especially verifiers using logarithmic space. In [10,12,18,20], the question of classifying the languages that have interactive proofs with various models of space-bounded verifiers is studied. The work of Dwork and Stockmeyer [12] and Kilian [29] consider the issue of when zero-knowledge interactive proof systems exist for systems with space bounded verifiers. The work of Cai et al. [7] considers the problem of designing untamperable benchmarks for other computers to follow. Their model considers the scenario of a resource-limited computer, which would like to ensure that a (very fast) computer has correctly computed benchmarks without taking any shortcuts. The main difference from this work is that in our model the verifier treats the prover as a black box and is only interested in what the answer is, rather than how (or how fast) it was computed.

2. The model

The model we describe below is based on probabilistically checkable proofs. Some features of this model are:

- It applies to function computations and decision, optimization, promise, approximation, and search problems.
- It allows proof systems in which the verifier is only convinced of the weaker assertion that a solution is approximately correct.
- It parametrizes the runtime of the verifier.
- It analyzes the runtime of the verifier implemented as a RAM machine in order to better understand the asymptotic complexity of the verifier.

We assume that the verifier is a RAM machine that has read access to an input tape, read access to the proof, read access to a source of random numbers, and read/write access to computation tapes. We assume that the verifier can read or write any word or number in any tape in constant time and perform arithmetic operations in constant time. This assumption is for simplicity of exposition since it affects the running time only by polylogarithmic factors. Let the random variable $\langle \Pi, \mathcal{V} \rangle (x, y)$ represent the output of the verifier $\mathcal{V}$ given a proof Π on input (x, y) when the random bits used by the verifier are chosen uniformly and independently.[1]

In the following, $\Delta(x, y)$ will be a distance function, where x is assumed to be the input, y is a candidate for $f(x)$ and $\Delta(x, y)$ gives an indication for how close y is to being correct. In some cases, $\Delta(x, y)$ may depend on the difference between $f(x)$ and y (as is typical when measuring the success of an approximation algorithm), in others $\Delta(x, y)$ may indicate the distance between x and the closest x' such that $f(x') = y$ (as is typically considered in measuring the success of a property testing algorithm). When Δ is used in the latter sense, we refer to the verifier as a *proof-assisted property tester*, which we describe in more detail below.

We now give definitions of approximate PCP models. Let $\Delta(\cdot, \cdot)$ be a distance function.

Definition 1 (*Approximate PCP*). A function f is said to have a $t(\epsilon, n)$*-time,* $s(\epsilon, n)$*-space,* ϵ*-approximate probabilistically checkable proof system* with distance function Δ if there is a randomized verifier $\mathcal{V}$ that, for all inputs $0 \leqslant \epsilon \leqslant 1$ and (x, y) of combined size n, runs in time $O(t(\epsilon, n))$ and

(1) if $\Delta(x, y) = 0$, then there is a proof Π of size $O(s(\epsilon, n))$ bits such that $\Pr[\langle \Pi, \mathcal{V} \rangle (x, y) = \mathsf{PASS}] \geqslant 3/4$ and

(2) if $\Delta(x, y) > \epsilon$, then for all proofs Π', $\Pr[\langle \Pi', \mathcal{V} \rangle (x, y) = \mathsf{FAIL}] \geqslant 3/4$.

Remarks.

(i) By running verifier $O(\log 1/\delta)$ times and taking the majority, one can obtain a confidence at least $1 - \delta$, for any $\delta > 0$. For simplicity, we construct verifiers for a constant δ.

[1] Note that the random bits can be used to determine which bits are read from the proof and thus can determine the output of the verifier.

(ii) The output of the verifier is not specified when $0 < \Delta(x, y) \leqslant \epsilon$.
(iii) The choice of the distance function Δ is problem-specific, and determines the ability to construct a proof system, as well as determining how interesting the proof system is. The usual definitions of probabilistically checkable proof systems for decision problems require that when $y = f(x)$, a correct proof can convince the verifier of that fact, and when $y \neq f(x)$, no proof can convince the verifier of the same. In our model, this is achieved by choosing $\Delta(\cdot, \cdot)$ such that $\Delta(x, y) > \epsilon$ whenever $y \neq f(x)$ and $\Delta(x, y) = 0$ when $y = f(x)$.
(iv) Δ need not be computable by the verifier. Thus, one can define probabilistically checkable proofs for promise problems, in which there are inputs for which the verifier is allowed to either pass or fail, by setting Δ to $\epsilon/2$ on those inputs. Independently of this work, Szegedy [40] has given a related formulation of probabilistically checkable proof systems in terms of three-valued logic that also applies to promise problems.
(v) We often omit the proof size $s(\epsilon, n)$ in our theorems.

For the special case when $\Delta(x, y)$ measures the distance between x and the "closest" x' such that $f(x') = y$, we call the proof system a $(\epsilon, t(\epsilon, n), s(\epsilon, n))$*-proof-assisted property tester*. Here, the definition of "closest" is usually in terms of the relative Hamming distance between x and x', i.e., the ratio of the Hamming distance between x and x' and the size of x. Thus, the verifier passes all x such that $f(x) = y$, and on the other hand, if the verifier passes (x, y), one can assume that there is an x' such that (1) x and x' are ϵ-close (according to the specified distance metric) and (2) $f(x') = y$. In the case of property testing, we will typically omit y from the parameter list to Δ. As before, $s(\epsilon, n)$ denotes the size of the proof (in bits) and $t(\epsilon, n)$ bounds the running time of the verifier.

We now give specific definitions for approximate lower and upper bound PCPs. All of these definitions are special cases of Definition 1.

Definition 2 (*Approximate lower (upper) bound PCP*)**.** A function f is said to have a $t(\epsilon, n)$*-time*, $s(\epsilon, n)$*-space*, ϵ*-approximate lower bound (resp. upper bound)* PCP if there is a randomized verifier $\mathcal{V}$ that, for all inputs $0 \leqslant \epsilon \leqslant 1$ and (x, y) of combined size n, runs in time $O(t(\epsilon, n))$ and
(1) if $y \leqslant f(x)$ (resp. $y \geqslant f(x)$), then there is a proof Π of size $O(s(\epsilon, n))$ bits such that $\Pr[\langle \Pi, \mathcal{V}\rangle(x, y) = \mathsf{PASS}] \geqslant 3/4$ and
(2) if $(1 - \epsilon)y > f(x)$ (resp. $(1 + \epsilon)y < f(x)$), then for all proofs Π', $\Pr[\langle \Pi', \mathcal{V}\rangle(x, y) = \mathsf{FAIL}] \geqslant 3/4$.

The multiplicative approximate lower and upper bound definitions correspond to setting $\Delta(x, y) = \max\{0, 1 - f(x)/y\}$ and $\Delta(x, y) = \max\{0, f(x)/y - 1\}$, respectively, in Definition 1.

Definition 3 (*Approximate additive lower (upper) bound PCP*)**.** A function f is said to have a $t(\epsilon, n)$*-time*, $s(\epsilon, n)$*-space*, ϵ*-approximate additive lower bound (resp. upper bound)* PCP if there is a randomized verifier $\mathcal{V}$ that, for all inputs $0 \leqslant \epsilon \leqslant 1$ and (x, y) of combined size n, runs in time $O(t(\epsilon, n))$ and
(1) if $y \leqslant f(x)$ (resp. $y \geqslant f(x)$), then there is a proof Π of size $O(s(\epsilon, n))$ bits such that $\Pr[\langle \Pi, \mathcal{V}\rangle(x, y) = \mathsf{PASS}] \geqslant 3/4$ and
(2) if $y > f(x) + \epsilon$ (resp. $y < f(x) - \epsilon$), then for all proofs Π', $\Pr[\langle \Pi', \mathcal{V}\rangle(x, y) = \mathsf{FAIL}] \geqslant 3/4$

The additive approximate lower and upper bound definitions correspond to setting $\Delta(x, y) = \max\{0, y - f(x)\}$ and $\Delta(x, y) = \max\{0, f(x) - y\}$, respectively, in Definition 1.

Notation. We use $x \in_R S$ to denote that x is chosen uniformly at random from a set S. We use $[n]$ for the set $\{1, \ldots, n\}$. We use b to denote the number of bits in a memory word and we assume all integer variables fit in a word.

For a function or property $f(x, \ldots)$, let $\Pi_f(x, \ldots)$ denote the proof and let $\mathcal{V}_f(\epsilon, \Pi)$ denote the verifier for this function/property checking a proof Π. The proof Π_f will consist of various components and data structures, each of which will be referred to using the record notation '$\Pi.\cdot$'. For example, if the proof Π has an array called T, then we use $\Pi.T$ to specify the verifier's access to this array in the proof.

3. Some basic building blocks

3.1. Multiset equality (permutation enforcement)

Given an input list $X = \langle x_1, \ldots, x_n \rangle$, many of our approximate PCPs require that the proof contain the list written in a different order $Y = \langle y_1, \ldots, y_n \rangle$ (for example, the sorted order). To be able to describe when X and Y above are (close to being) permutations of each other, we need to extend set intersection to multisets: an element i occurs exactly k times in $X \cap Y$ if and only if it occurs exactly k times in one of X, Y, and at least k times in the other. For this problem we would like the verifier to be able to ensure that $|X \cap Y| \geqslant (1 - \epsilon)n$. In particular, the verifier should be able to access elements from Y while ensuring that each accessed element corresponds to a unique element in X. The difficulty comes from the possibility that the elements in each list are not necessarily distinct. One would like to prevent the possibility that an x_i from X appears more than once in Y, or that two equivalent elements $x_i = x_j$ in X are replaced by only one element in Y. Without any aid, the verifier requires $\Omega(\sqrt{n})$ time to ensure that $|X \cap Y| \geqslant (1 - \epsilon)n$ [14]. Here we show that it can be done in $O(1/\epsilon)$ time by requiring the proof to be written in a special format. The special format consists of the *permutation enforcer* – two arrays T_1, T_2 of length n, where the contents of location i in T_1 contains a pointer to the location of x_i in Y and the contents of location i in T_2 contains a pointer to the location of y_i in X.

Given two multisets X, Y, let $\Delta(\{X, Y\}) = 1 - |X \cap Y|/n$. Below we give the proof-assisted property tester for the problem.

```
The proof $\Pi_{\text{PE}}(X, Y, n)$
  For permutation $\sigma$ such that $x_i = y_{\sigma(i)}$ for all $i$:
  $T_1[i] = \sigma(i)$ for all $i$;
  $T_2[i] = \sigma^{-1}(i)$ for all $i$;

The verifier $\mathcal{V}_{\text{PE}}(X, Y, n, \epsilon, \Pi)$:
  Repeat $O(1/\epsilon)$ times:
    Choose $i \in_R [n]$
    If $x_i \neq y_{\Pi.T_1[i]}$ or $\Pi.T_2[\Pi.T_1[i]] \neq i$ output FAIL
  Output PASS
```

Theorem 4. *Given two multisets of size n, there is an* $(\epsilon, 1/\epsilon, n\log n)$*-proof-assisted property tester for multiset equality.*

Proof. Call index i *good* if $x_i = y_{\Pi.T_1[i]}$ and $\Pi.T_2[\Pi.T_1[i]] = i$; i is *bad* otherwise. The set of bad indices is exactly the set of indices which cause the verifier to fail.

If $\Delta(\{X,Y\}) = 0$, i.e., if $X = Y$, then there is a permutation σ for which $x_i = y_{\sigma(i)}$ for all i. Thus, using σ to define T_1, T_2 results in all indices being good, and the verifier will always output PASS.

To show that if $\Delta(\{X,Y\}) \geqslant \epsilon$ the verifier outputs FAIL with high probability, we note that the number of good indices presented by any proof is upper bounded by $|X \cap Y|$. Conversely, since $\Delta(\{X,Y\}) \geqslant \epsilon$, there must be fewer than $(1-\epsilon)n$ good indices; it is easy to see that $O(1/\epsilon)$ trials suffice for the verifier to come upon an index that is not good and output FAIL with probability at least 3/4. □

3.1.1. Set intersection

Similar ideas can be used to obtain additive approximate upper and lower bound PCPs for set intersection. One application of these PCPs is to proving bounds on the sizes of unions and intersections of databases queries.

The set intersection problem is: given sets A_0 and A_1 of n elements coming from a domain D, and parameter ρ, is $|A_0 \cap A_1|$ approximately ρn? We assume that the sets are represented by an array of size n, where the ith location contains the ith element of the set. Let $f(A_0, A_1) = |A_0 \cap A_1|/n$. The verifier will use the proof in order to distinguish the case where $\rho - \epsilon/2 \leqslant f(A_0,A_1) \leqslant \rho + \epsilon/2$ from the case where $f(A_0,A_1) < \rho - \epsilon$ or $f(A_0,A_1) > \rho + \epsilon$.

Without the aid of a proof, the verifier requires $\Omega(\sqrt{n})$ time [14]. The lower bound protocol of Goldwasser and Sipser [27] can be adapted to this setting to get multiplicative approximations of a lower bound on $|A_0 \cap A_1|$, but we know of no such way to get a multiplicative approximation for the upper bound using the methods of Fortnow [17], since they require a fast method of generating a random element of $A_0 \cap A_1$. Our techniques can be viewed as special cases of the techniques in [17,27], where the identity function is used in place of a hash function. The approximate PCP is as follows:

```
The proof $\Pi_{\text{SET}}(A_0, A_1, \rho)$:
  $T_0$ s.t. $\forall x \in D$, $T_0[x] = i$ if $A_0[i] = x$ and $T_0[x] = 0$ otherwise
  $T_1$ s.t. $\forall x \in D$, $T_1[x] = i$ if $A_1[i] = x$ and $T_1[x] = 0$ otherwise

The verifier $\mathcal{V}_{\text{SET}}(A_0, A_1, \rho, \epsilon, \Pi)$:
  Repeat $m = O(1/\epsilon^2)$ times:
    Choose $c \in_R \{0,1\}$
    Choose $x \in_R A_c$
    Let $i_c = \Pi.T_c[x]$, $i_{1-c} = \Pi.T_{1-c}[x]$
    If $A_c[i_c] \neq x$ or $(i_{1-c} \neq 0$ and $A_{1-c}[i_{1-c}] \neq x)$ then output FAIL.
    If $i_{1-c} \neq 0$ then let $k = k + 1$
  If $\rho - 3\epsilon/4 \leqslant k/m \leqslant \rho + 3\epsilon/4$ then output PASS
  Output FAIL
```

Theorem 5. *There is a* $(1/\epsilon^2)$*-time* ϵn*-additive approximate upper and lower bound PCP for two set intersection.*

Proof. We will show something stronger than the claimed theorem, namely we show that the above protocol actually checks that the estimate ρ is correct to within an additive error of $\epsilon/4$. This will prove that the protocol is both an upper bound and a lower bound PCP.

Let $x \in D$ be *good* if $\Pi.T_0[x] \neq 0, \Pi.T_1[x] \neq 0, A_0[\Pi.T_0[x]] = x$ and $A_1[\Pi.T_1[x]] = x$. It is easy to see that x is good if and only if $x \in A_0 \cap A_1$. The verifier uses the quantity $(k/m)n$ as an estimate of the number of good elements, i.e., it estimates $|A_0 \cap A_1|$. Using Chernoff bounds [9], it is easy to see that, with constant probability, $|A_0 \cap A_1|/n$ and k/m are off by an additive factor of at most $\epsilon/4$. For the remainder of the proof, we will assume that this event has happened, i.e., $||A_0 \cap A_1|/n - (k/m)| \leqslant \epsilon/4$.

Suppose $|A_0 \cap A_1|/n \leqslant \rho + \epsilon/2$. Then, $k/m \leqslant |A_0 \cap A_1|/n + \epsilon/4 \leqslant (\rho + \epsilon/2) + \epsilon/4 = \rho + 3\epsilon/4$ and so the verifier will output PASS. A similar argument can be made for the case when $|A_0 \cap A_1| /n > \rho - \epsilon/2$.

Conversely, suppose $|A_0 \cap A_1|/n > \rho + \epsilon$. Then, $k/m \geqslant |A_0 \cap A_1|/n - \epsilon/4 > (\rho + \epsilon) - \epsilon/4 = \rho + 3\epsilon/4$ and so the verifier will output FAIL. A similar argument can be made for the case when $|A_0 \cap A_1|/n < \rho - \epsilon$. □

Note that, in the above, even though the arrays T_i are large, they are referenced only indirectly as a result of sampling A_0 or A_1, and the running time of the protocol is not adversely affected.

In general, if A_0 and A_1 are sets of different, but known sizes, using a variant of the above approximate PCP, we can obtain upper and lower bounds on $|A_0 \cap A_1|/(|A_0| + |A_1|)$. Also, note that using inclusion–exclusion, these methods can be used to estimate the size of two set union as well.

This also gives approximate PCPs for checking, given $A_0, \ldots, A_k$ if $|\bigcap_{i=1}^{k} A_i|$ is large: the verifier picks $i \in_R [k]$ and then $x \in_R A_i$ and queries the location corresponding to i,x in the proof. At the location corresponding to i,x, the proof contains k pointers to the locations of x in each of A_i's. The verifier ensures that these pointers are valid. The analysis is similar to that of Theorem 5.

3.2. Element distinctness

Given an input list $X = \langle x_1, \ldots, x_n \rangle$, it is often useful for the verifier to ensure that the x_i's are distinct. Here we give a proof Π_{ED} and a $O(1/\epsilon)$ time verifier $\mathcal{V}_{\text{ED}}$, which uses Π_{ED} to ensure that the number of distinct elements in X is at least $(1 - \epsilon)n$. Without the aid of a proof, the verifier requires $\Omega(\sqrt{n})$ time to determine the same [14]. Our method can be viewed as a simplification of the protocols given by Fortnow [17] and Goldreich et al. [24], in our setting. The protocol of Fortnow [17] in the interactive proof setting allows a prover to convince a verifier of an upper bound on the size of a set (this protocol can be adapted for use in the probabilistically checkable proof setting). Interestingly, we use a similar idea here in the proof-assisted property tester setting to give a lower bound on the size of a set. Our proof Π_{ED} consists of an array A such that $A[x]$ has a pointer to the location of x in X.

Let $\Delta(X) = 1 - |X|/n$ where $|X|$ denotes the number of distinct elements in X. We now show:

```
The proof $\Pi_{\text{ED}}(X, n)$:
  $A$ s.t. $\forall x \in D$, $A[x] = i$ if $x = x_i$ for $x_i \in X$
    and $A[x] = 0$ otherwise

The verifier $\mathcal{V}_{\text{ED}}(X, n, \epsilon, \Pi)$:
  Repeat $O(1/\epsilon)$ times:
    Choose $i \in_R [n]$
    If $i \neq \Pi.A[x_i]$ output FAIL
  Output PASS
```

Theorem 6. *Let Δ be defined as above. Given two multisets of size n, the element distinctness problem has*:

(a) *an $(\epsilon, 1/\epsilon, n + U \log n)$-proof-assisted property tester where U is an upper bound on the value of x_i's, and*

(b) *an $(\epsilon, (1/\epsilon) \log n, n \log n)$-proof-assisted property tester.*

Proof.

(a) If $\Delta(X) = 0$, i.e., if the multiset X consists of distinct elements, then the proof Π_{ED} will make the verifier $\mathcal{V}_{\text{ED}}$ accept. If $\Delta(X) > \epsilon$, i.e., if the number of distinct elements in X is less than $(1 - \epsilon)n$, then the probability that the verifier chooses an i corresponding to a non-distinct element is at least ϵ, and if x_i is not distinct, the probability that $j = i$ is at most 1/2. Thus, there is a constant c such that after c/ϵ trials, the verifier will output FAIL with probability at least 3/4.

(b) To make the size of the proof independent of U, we construct a new proof $\Pi_{\text{ED}'}$ in a different format. The idea is to compress the proof in (a) by only storing the nonzero elements of A in an ordered list. The proof $\Pi_{\text{ED}'}$ contains the answers to the queries as a list of n ordered pairs containing each input element and its location in the input list (x_i, j) in order sorted by the value of x_i. The verifier then performs a binary search to find (x_i, j) based on the keyword x_i and checks if $j = i$. The rest of the analysis is as in (a) except that the verifier runs in time $O((1/\epsilon) \log n)$. □

The above element distinctness protocol can be applied to give an efficient proof assisted property tester for the following problem: Given an $n \times n$ operation table for $\circ$, is $\circ$ an associative operation? We would like to output PASS if $\circ$ is associative and return FAIL if at least ϵ fraction of the table entries need to be changed in order to turn $\circ$ into an associative operation. The best known property tester for associativity runs in $O(n^{1.5}\text{poly}(\log n))$ time [14]. One main bottleneck in that test is that we need to look at the operation table and ensure that all columns and all rows are mostly distinct. For each column/row, this requires $\Omega(\sqrt{n})$ time without the aid of the proof. Using the above result, testing that a row or column is mostly distinct can be done in constant time and thus one can give an proof-assisted property tester for associativity whose runtime is $O(n\text{poly}(\log n))$.

3.3. Lower bounds on the size of a set

Given a list L, it is nontrivial to deduce the number of distinct elements in L. Let S_L denote the set of distinct elements in L and let $f(L) = |S_L|$. Suppose the verifier can easily determine whether a b-bit element x is in S_L. (For example, the verifier may be given a list of all possible b-bit values along with pointers, if any, to their location in L, which it could utilize to check in constant time whether $x \in S_L$ for any b-bit x.) Then the verifier could estimate $f(L)/2^b$ to within a multiplicative error of ϵ by sampling: choose a random b-bit element x and check if x belongs to the list L. This requires $\Omega(2^b/(\epsilon f(L)))$ samples [8,11].

The method we present below is significantly more efficient; it is simple, fast, and has one-sided error. We note that there are protocols for lower bounding set size due to Goldwasser and Siper [27] and Furer et al. [21] that can also be performed in the approximate PCP setting (the former protocol has 2-sided error and the latter is slightly less efficient than the one given here). In our applications of these protocols in Sections 3.4 and 4.1, any one of the three can be used interchangeably.

Let s be the claimed number of distinct elements in L. In our construction, the proof Π_{SZ} contains an array A consisting of the s distinct elements in L. We use the verifier $\mathcal{V}_{\text{ED}}$ on the proof $\Pi_{\text{ED}}(A)$ to check the distinctness of the elements in A, such that this verifier has probability of error at most 1/8. To check in addition that the elements of A indeed come from L, we assume that we have a membership oracle pair, consisting of Π_{MEM} and $\mathcal{V}_{\text{MEM}}$. The proof Π_{MEM} creates data structures for the list L so that the verifier $\mathcal{V}_{\text{MEM}}$ can efficiently check if a given x belongs to L. We assume that the membership oracle proof Π_{MEM} has access to the proof Π_{SZ}. The reason for introducing the oracle pair is that we will be using the same approximate PCP for set size, but with different assumptions on the input, that lead to different membership oracle pairs in later constructions.

```
The proof $\Pi_{\text{SZ}}(L, s, \Pi_{\text{MEM}})$:
  $A[1, \ldots, s]$ containing distinct elements of $L$;
  $P_1 = \Pi_{\text{ED}}(A)$;
  $P_2 = \Pi_{\text{MEM}}(L)$.

The verifier $\mathcal{V}_{\text{SZ}}(L, s, \epsilon, \Pi, \mathcal{V}_{\text{MEM}})$:
  Run $\mathcal{V}_{\text{ED}}(\epsilon/2, \Pi.P_1)$
  Repeat $O(1/\epsilon)$ times:
    Choose $i \in_R [s]$
    Run $\mathcal{V}_{\text{MEM}}(A[i], \Pi.P_2)$
  Output PASS
```

We now show how to construct the membership oracle pairs for the simplest case of set size. We give two schemes: a simple scheme that works when the domain is bounded and a more sophisticated scheme that uses pointers.

(1) In the first scheme, the proof $\Pi_{\text{MEM}}(L)$ consists of an array T of pointers for all elements in the domain where $T[x]$ contains the location in L that contains x. The verifier $\mathcal{V}_{\text{MEM}}(x, \Pi)$ can just check if $L[\Pi.T[x]] = x$.

(2) In the second scheme, we use pointers to make the size of the data structures independent of the domain size (but still dependent on $|L|$). For this scheme, the proof is assumed to know the array A from the proof Π_{SZ} and the verifier is assumed to invoke the oracle $\mathcal{V}_{\text{MEM}}$ with i instead of $A[i]$.

The proof Π_{MEM} contains an array P, consisting of elements of A as well as a pointer from each such element to its location in L. For instance, if $L = \{4, 4, 2, 3, 3, 1, 6, 6, 6, 6\}$, then $A = \{4, 2, 3, 1, 6\}$ and the proof Π_{MEM} consists of $P = \{\langle 4, 1\rangle, \langle 2, 3\rangle, \langle 3, 4\rangle, \langle 1, 6\rangle, \langle 6, 7\rangle\}$. The verifier $\mathcal{V}_{\text{MEM}}$, checks given a position i, whether $A[i]$ indeed is present in L by following a back pointer from $P[i]$ in constant time.

```
The proof $\Pi_{\text{MEM}}(L)$:
  $P$ s.t. for $1 \leqslant i \leqslant m$, $P[i] = \langle A[i], j \rangle$ where $L[j] = A[i]$.

The verifier $\mathcal{V}_{\text{MEM}}(L, i, \Pi)$:
  Let $\langle a, j \rangle = \Pi.P[i]$
  If $L[j] \neq a$ or $A[i] \neq a$ output FAIL
```

Theorem 7. *There is a* $(1/\epsilon)$-*time*, $O(n \log n)$-*space*, ϵ-*approximate lower bound PCP for the size of a set represented by a list.*

Proof. Let s be the claimed number of distinct elements in L. If L is such that $f(L) \geqslant s$, then there is a proof that makes the verifier always output PASS. Conversely, if either the fraction of distinct elements in A is smaller than $(1 - \epsilon/2)$ or at least $\epsilon/2$ fraction of the elements in A are not in L, then the verifier is likely to fail. Thus, treating A and L as sets, the verifier is only likely to pass if $|A \cap L| \geqslant (1 - \epsilon)s$. □

3.4. Lower bounds on sums

Given a list of positive integers $x = \langle x_1, \ldots, x_n \rangle$, we show how the verifier can be convinced of a good approximation to a lower bound s on $f(x) = \sum_i^n x_i$. Without any aid, the verifier requires $\Omega(n)$ time to estimate the lower bound, since it is possible that all but one of the x_i's are 0. We give two methods by which the verifier, with the aid of a proof, can be convinced that the sum is at least $(1 - \epsilon)s$. The first method requires only that the verifier use constant time but requires a very large proof size (proportional to the magnitude of the sum). The latter requires that the verifier spend $O(\log B)$ time, where B is an upper bound on the x_i's (since we assume x_i fits in a memory word, $B < 2^b$) but requires a proof of smaller size.

Using approximate lower bound PCPs. Consider the set $S = \{(i, j) \mid 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant x_i\}$ (if $x_i = 0$ then there will be no j such that $(i, j) \in S$) whose cardinality is $\sum_{i=1}^n x_i$. Note that one can construct membership oracle pairs for membership in S: the proof $\Pi_{\text{MEM}}(S)$ is empty and verifier $\mathcal{V}_{\text{MEM}}((i, j), \Pi)$ just checks (in constant time) if $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant x_i$. Using this membership oracle pair with the proof Π_{SZ} and the verifier $\mathcal{V}_{\text{SZ}}$ from the previous section, we can get a lower bound PCP for the size of S where the running time of the verifier is $O(1/\epsilon)$.

Theorem 8. *There is a* $(1/\epsilon)$-*time* $(nB \log(nB))$-*space* ϵ-*approximate lower bound PCP for the sum of n positive integers, each within the range* $[B]$.

Grouping elements by size. In this method, the verifier uses random sampling to estimate the sum. Since the number of samples required to get good estimates depends on the variance of the sample, the proof Π_{SUM} presents the x_i's in groups for which the variance is small: the x_i's in Π_{SUM} are grouped such that the ith group contains all x_i whose weights are between $B/2^i$ and $B/2^{i+1}$, and each group is represented by a separate array along with the size. Since we assume integer weights, there are at most $1 + \log B$ such groups. A cheating proof could make the sum look larger than it is by including additional large elements in the arrays that are not present in the original list of the x_i's, or by listing large x_i's multiple times. In order to protect against this, we use the permutation enforcer as the membership oracle pair Π_{PE} and $\mathcal{V}_{\text{PE}}$. The proof Π_{SUM} contains the permutation enforcer proof $\Pi_{\text{SUM}}.\Pi_{\text{PE}}$. The verifier $\mathcal{V}_{\text{SUM}}$ uses the permutation enforcer verifier $\Pi_{\text{SUM}}.\mathcal{V}_{\text{PE}}$ to ensure that each element sampled comes uniquely from the original set of x_i's. Suppose the verifier chooses element y_j in one of the groups. Say that y_j is *good* if its weight is consistent with its group and the permutation enforcer Π_{PE} is consistent, i.e., $\Pi_{\text{PE}}.T_1[\Pi_{\text{PE}}.T_2[j]] = j$ and $x_i = y_{\Pi_{\text{PE}}.T_1[i]}$. Let $G = \{j \mid y_j \text{ is good}\}$. Then $\sum_{j\in G} y_j \leqslant \sum_i x_i$. The verifier uses sampling to lower bound $\sum_{j\in G} y_j$. To do this, suppose the ith group has n_i elements. Then the verifier picks $O((1/\epsilon)\log\log B)$ elements from the ith group, checks that they are good, and sets S_i to be their average multiplied by n_i. Finally, the verifier computes an estimate $\tilde{s} = \sum_i S_i$ for the sum, and outputs PASS if and only if $s(1-\epsilon/2) \leqslant \tilde{s}$.

Theorem 9. *There is a $((1/\epsilon)\log\log B)$-time, $(n\log n\log B)$-space, ϵ-approximate lower bound PCP for the sum of n integers, each of which is in the range $[B]$.*

Proof. Let s be a lower bound on $f(x)$. Let $\tilde{s} = \sum_{j=1}^{1+\log B} S_j$ be the estimate of $f(x)$ computed by the verifier. By [11] we know that a sampling scheme can obtain an $\epsilon/2$-approximation for the sum of the good elements in the jth group with probability at least $3/(4\log B)$. Thus, $f(x)(1-\epsilon/2) \leqslant \tilde{s} \leqslant f(x)(1+\epsilon/2)$ with probability at least 3/4.

First consider the case where $s \leqslant f(x)$. It is easy to see that, for $\epsilon \leqslant 1$, with probability at least 3/4, there is a proof such that the verifier will output PASS. Now assume that $s(1-\epsilon) > f(x)$. We have from the goodness of our approximation that $\tilde{s} \leqslant (1+\epsilon/2)f(x)$ with probability at least 3/4. Combining the two inequalities above, we have that $s(1-\epsilon/2) > \tilde{s}$, in which case the verifier returns FAIL. The total running time is $O((1/\epsilon)\log\log B)$. □

4. PCPs for optimization and graph problems

4.1. Constraint satisfaction problems

A warmup: Lower bounds on the cut size. We give a simple approximate PCP where the verifier can be quickly convinced by a proof that a given graph has a large cut. The main idea is for the proof to present the cut, and then prove that the cut is indeed large by using the approximate PCP for proving a lower bound on the size of a set.

We first describe the approximate lower bound PCP for cut size in an unweighted graph $G = (V, E)$. Given a cut $[S, V\backslash S]$, for each vertex v, let D be an array such that $D[v] = 1$ if $v \in S$ and $D[v] = 0$ otherwise. Assume that G is represented as an adjacency matrix so that membership in E can be determined in constant time for any vertex pair (u, v). Let $C = E \cap \{(u,v) \mid D[u] \neq D[v]\}$ denote the set of edges across the cut. Note that the membership oracle pair for C is easy to construct: the proof Π_{MEM} is empty and the verifier $\mathcal{V}_{\text{MEM}}((i,j),\Pi)$ checks first whether $(i,j) \in E$, and then verifies that $\Pi.D[i] \neq \Pi.D[j]$ in constant time. Together with the approximate lower bound PCP for set size from Section 3.3, we obtain a $(1/\epsilon)$-time ϵ-approximate lower bound PCP for maximum cut.

```
**The proof** $\Pi_{\text{CUT}}(G,c)$:
  For a cut $S$ of size $c$:
    $D$ s.t. $\forall i \in V$, $D[i]=0$ if $i \in S$; $D[i]=1$ if $i \notin S$;
  $P_1 = \Pi_{\text{SZ}}(C = E \cap \{(u,v) \mid D[u] \neq D[v]\}, c, \Pi_{\text{MEM}})$.

**The verifier** $\mathcal{V}_{\text{CUT}}(G,c,\epsilon,\Pi)$:
  Run $\mathcal{V}_{\text{SZ}}(C = E \cap \{(u,v) \mid D[u] \neq D[v]\}, c, \epsilon, \Pi.P_1, \mathcal{V}_{\text{MEM}})$
```

The weighted case may be treated by using an approximate lower bound PCP for the sum $\sum_{(u,v)\in C} w(u,v)$, where $w(u,v)$ is the weight of edge (u,v), and has the same complexity as the unweighted one.

When the input graph is given in terms of its adjacency matrix, obtaining a sublinear proof to convince the verifier of a multiplicatively approximate upper bound on the size of a given cut is not possible, since the verifier requires $\Omega(n^2)$ time to distinguish between a cut size of 0 and 1.

Estimating the size of the maximum cut is related to the problem of testing whether a graph is bipartite. In particular, the number of edges minus the size of the maximum cut is exactly the number of edges that need to be removed from the graph in order to make it bipartite. In the adjacency matrix model, a property tester for bipartiteness should pass graphs G that are bipartite and fail graphs G for which more than ϵn^2 edges need to be removed in order to make G bipartite. A poly$(1/\epsilon)$ time algorithm for testing bipartiteness was given in [23]. The above model does not yield interesting results for sparse graphs, as every sparse graph is such that at most ϵn^2 edges need to be removed in order to make it bipartite. Thus, Goldreich and Ron [25,26] consider a property testing model for sparse graphs which has the following behavior: if the graph has bounded degree d and is represented in the adjacency list representation, the property tester must now pass bipartite graphs and fail graphs for which ϵdn edges need to be removed in order to make the graph bipartite. An $O(\sqrt{n}\,\text{poly}(\log n))$ time algorithm was given in [26] that satisfies the new requirements. It is also known that $\Omega(\sqrt{n})$ time is required to solve this problem [25]. The above approximate PCP for lower bounding the cut size can be used to give an $(\epsilon, \text{poly}(1/\epsilon))$-proof-assisted property tester for the bipartiteness of sparse graphs. If the input graph is given in a format for which the verifier can easily choose a random edge, then the problem is even easier: by requiring Π to write down the side of the cut that each vertex is on (say, the color of each vertex), a poly$(1/\epsilon)$ verifier can ensure that most edges cross the cut (or have endpoints with different colors).

Maximum constraint satisfaction problems. Constraint satisfaction problems (CSP) [28,38] refer to a class of problems that can be represented as follows: Define a set of *constraint functions* $f_1, \ldots, f_\ell : \{0,1\}^k \to \{0,1\}$ such that f_i is satisfied by $x \in \{0,1\}^k$ if $f_i(x)=1$. A *constraint application* of f_i to Boolean variables $x_1, \ldots, x_n$ is an ordered pair $\langle f_i, (a_1, \ldots, a_k)\rangle$, which is satisfied if $f_i(x_{a_1}, \ldots, x_{a_k}) = 1$. We assume constraints can be evaluated in $O(k)$ time. On input a collection of constraint applications on Boolean variables $x_1, \ldots, x_n$, the *Max CSP* problem is to find a Boolean setting of the x_i's such that the number of satisfied constraints is maximized. In the case that the input also includes weights on the constraint applications, the *Weighted Max CSP* problem involves finding a setting of the x_i's that maximizes the sum of the weights of the satisfied constraints. The Max SAT problem and the Max Cut problem can both be cast as maximum constraint satisfaction problems. We show below how to construct an approximate lower bound PCP for the general Weighted Max CSP Problem.

Theorem 10. *Let n be the number of variables and let k be the maximum size of constraints for a Weighted Max CSP problem Γ. Then there is a (k/ϵ)-time, ϵ-approximate lower bound PCP for Γ.*

Proof. Let v be the purported value to the Weighted Max CSP problem and let F be the subset of constraints that are satisfied. The proof Π_{CSP} to lower bound this value by at least $(1 - \epsilon)v$ consists of two parts.

(1) The 0/1 settings of the x_i's.
(2) A proof $\Pi_{\text{SUM}}(v, F)$ for showing an approximate lower bound on the sum of the weights of the constraints in F.

This combined proof can be used to convince the verifier that the sum of the weights of the satisfied constraints is at least $(1 - \epsilon)v$. The verifier, while running the permutation enforcer for checking the set size proof, also checks that the settings of the x_i's in the proof satisfy the constraints in F. It is clear that if all the constraints in F are satisfied, and the weights add up to v, the verifier always returns PASS. If the total weight is less than $(1 - \epsilon)v$, then using the protocols for lower bounding set sizes and sums (see Theorems 7, 8, and 9), one can see that the verifier returns FAIL with probability at least 3/4. Checking each constraint takes $O(k)$ time, and the techniques for bounding sums require that the verifier look at $O(1/\epsilon)$ constraints. □

Min Ones CSPs. Min Ones CSP involves finding a setting of the x_i's to satisfy all of the given constraints while minimizing the number of x_i's set to 1. The value of the optimization problem is $f(x) = |\{i \mid x_i = 1\}|$, the number of x_i's set to 1. To illustrate, we present an example of Min Ones CSP, the vertex cover problem on a graph of maximum degree d. This problem is NP-complete for any $d \geqslant 3$. Given graph $G = (V, E)$ of degree at most d with $|V| = n$, $|E| = m$, and a bound B, one would like to know whether there is a vertex cover $C \subseteq V$ such that $|C| \leqslant B$.

We present an ϵ-approximate upper bound PCP for vertex cover, assuming the graph is presented in such a way that a uniformly distributed edge can be chosen in constant time. The proof represents the vertex cover by writing array C of size at most B, which contains all the vertices in the cover. Then, array P contains, for each edge, a pointer to a vertex in C that covers that edge. The verifier chooses $O(1/\epsilon)$ edges and using P, verifies that each edge is covered by the vertex cover defined by C. The verifier outputs FAIL if some edge is not covered. We give a more precise description below.

The proof $\Pi_{\text{VC}}(E, B)$:
```
  $C = \langle c_1, \ldots, c_B \rangle$ ;
  $P$ s.t. $\forall e = (u, v) \in E, P[(u, v)] = \langle i, j \rangle$ ,
    where $c_i$ covers $u$ and $c_j$ covers $v$.
```

The verifier $\mathcal{V}_{\text{VC}}(E, B, \epsilon, \Pi)$:
```
  Repeat $O(d/\epsilon)$ times:
    Choose $e = (u, v) \in_R E$
    Let $\langle i, j \rangle = \Pi.P[e]$
    If $((u, \Pi.c_i) \notin E)$or $((v, \Pi.c_j) \notin E)$ output FAIL
  Output PASS
```

We now show the following theorem.

Theorem 11. *Let G be a graph of degree at most d represented in such a way that a uniformly distributed edge can be chosen in constant time. Then, there is a* (d/ϵ)*-time,* $\mathrm{O}(B\log n + m\log n)$*-space,* ϵ*-approximate upper bound PCP for vertex cover on G, where B is the claimed size of the vertex cover.*

Proof. Let $f(G)$ be the size of the smallest vertex cover for graph G. If there is a vertex cover of size at most B, then there is a proof such that $\mathcal{V}_{\text{VC}}$ will always pass. If there is no vertex cover of size smaller than $B(1+\epsilon)$, then consider any set $V' \subseteq V$ of vertices such that $|V'| = B$. Let u be the number of edges not covered by V'. Then, $u \geqslant f(G) - B > f(G) - f(G)/(1+\epsilon) = f(G)\epsilon/(1+\epsilon) \geqslant f(G)\epsilon/2$, for $0 < \epsilon \leqslant 1$. Note that, since the maximum degree in the graph is d, $m/d \leqslant f(G)$. Then, $u > m\epsilon/(2d)$. As a result, the probability that an uncovered edge is chosen at any iteration in the above algorithm is $u/m \geqslant \epsilon/(2d)$. Thus the verifier is likely to find at least one edge that is not covered and output FAIL. □

A similar approach can be used for dominating set and set cover problems with bounded subset size of at least 3, which are also NP-complete.

Consider a slightly different problem, which we refer to as *Max B-Ones CSP*, where the value of the optimization problem is the number of constraints that can be simultaneously satisfied by a setting that only has B variables set to one. We design an ϵ-approximate lower bound PCP for this problem as follows. The proof contains an array X of size n, and another array C of size B; C contains a list of those variables that are set to 1. The ith entry of X contains $\langle j, p\rangle$, where j is the 0/1 setting for the variable x_i. If this setting is 1, then p points to the location of x in C, i.e., $p = j$ where $C[j] = i$. For a given constraint, the verifier can check that it indeed evaluates to 1 with the setting defined by X. In doing this, if it encounters a variable x_i that is set to 1 in X, it uses the the pointer p to check if $C[p] = i$. This guarantees that at most B variables are set to 1. Then, the approximate PCP for lower bounding the size of a set can be used to allow the verifier to ensure that at least $1-\epsilon$ fraction of the constraints are satisfied with the given setting. Thus the following can be achieved.

Theorem 12. *Let n be the number of variables and let k be the maximum size of a constraint for a Max B-Ones CSP problem. Then there is a* $\mathrm{O}(k/\epsilon)$*-time,* ϵ*-approximate lower bound PCP for Max B-Ones CSP.*

4.2. Constraint enforcement: approximate PCPs

We have seen that designing approximate lower bound PCPs seems much easier than designing upper bound counterparts. In this section we show that a proof can convince a verifier that a good solution to an optimization problem satisfies certain types of upper bound constraints. We first apply our technique to approximations for *t-sparse fractional packing problems* and then show how the technique can be used for other approximation problems.

Fractional packing problems. Fractional packing problems are a class of linear programming problems defined by Plotkin et al. [35]. We consider a *sparse* version of the problem where we are given $a_1, \ldots, a_n \geqslant 0$ and $b_{11}, \ldots, b_{nm} \geqslant 0$, such that for each i, at most t of the b_{ij}'s are nonzero (we refer to t as the *sparsity* of the problem). Let OPT be the solution to the following maximization problem: $\max\{\sum_{i=1}^{n} a_i x_i\}$ subject to $x_i \geqslant 0$ and the m constraints $\forall j \in [m], \sum_i b_{ij} x_i \leqslant c_j$. Since the b_{ij}'s are sparse, we assume that for each variable x_i, there is a list S_i of j such that $b_{ij} > 0$. (We assume this for convenience in presenting our approximate PCP. As long as there is an easy way to find all nonzero b_{ij}'s for any given i, other ways to represent the sparse data can be used.) We assume that all a_i's, b_{ij}'s, c_j's, and x_i's can be represented in a word in memory.

Given a solution of value OPT, we construct a proof that can convince a verifier that the value is at least $(1-\epsilon)$OPT and that the solution satisfies all of the constraints. To this end, we give a *approximate* PCP *for constraint enforcement*. All of our results apply to the case when there is a solution of value v (which is not necessarily optimal) and the proof can convince the verifier that the value is at least $(1-\epsilon)v$.

4.2.1. Constraint enforcement: unweighted version

In order to describe the approximate PCP for constraint enforcement, we begin with the simpler case of *unweighted fractional packing problems*, in which all the a_i's and b_{ij}'s are 1 or 0, and each x_i is further constrained to be either 1 or 0. Note that $b_{ij} x_i \in \{0,1\}$. The verifier must ensure that there are a large number of x_i's that are set to 1 such that they do not violate any of the constraints.

The proof Π_{CES} is the following *constraint enforcement structure* that consists of three parts:

(1) An array X of length n such that the i-th entry is the value of x_i.

(2) For the jth constraint, a list C_j of the x_i's that are allocated "space" in the constraint (i.e., those x_i for which $b_{ij} x_i = 1$). More specifically, this part of the proof consists of *constraint arrays* $C_1, \ldots, C_m$, where C_j is of length c_j. For every x_i such that $b_{ij} > 0$ and such that x_i is set to 1 in the optimal solution, there is a location ℓ such that $C_j[\ell] = i$. If space is allocated in C_j for each x_i such that $b_{ij} x_i = 1$, since the size of C_j is c_j, the capacity constraints are not violated.

(3) For each i, pointers to the locations in the constraint arrays in which x_i is allocated space (i.e., $b_{ij} = x_i = 1$), so that for each x_i set to 1, the verifier can ensure that it is allocated space in each constraint that contains a term $b_{ij} x_i$ with $b_{ij} > 0$. More specifically, a modification of permutation enforcement is used: the proof contains an array T with n entries of total size at most t, such that $T[i] = (\langle j_1, \ell_1 \rangle, \ldots, \langle j_t, \ell_t \rangle)$ where $\langle j_a, \ell_a \rangle \in T[i]$ whenever x_i is present in constraint j_a (i.e., $b_{ij_a} > 0$) and ℓ_a is that location in C_{j_a} such that $C_{j_a}[\ell_a] = i$.

Fig. 1 shows the constraint enforcement structure used for the following problem: Maximize $x_1 + x_2 + x_3 + x_4$ subject to constraint 0 : $x_1, x_2, x_3, x_4 \in \{0,1\}$, constraint 1: $x_1 + x_2 \leqslant 1$, constraint 2: $x_2 + x_3 + x_4 \leqslant 2$, and constraint 3: $x_1 + x_3 \leqslant 1$. The solution setting $x_1 = x_4 = 1$ and $x_2 = x_3 = 0$ has value 2.

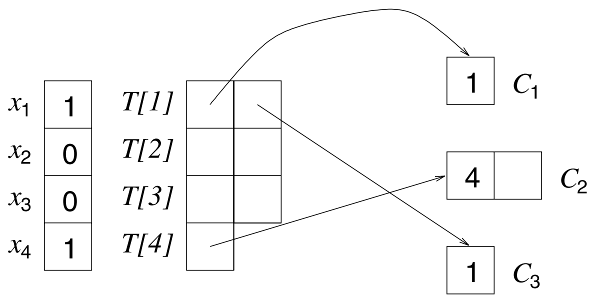

Fig. 1. Π_{CES} for the unweighted case.

Note that both T and C are arrays where the entries are of variable size. Using common data structures, we can assume that the verifier is able to access an arbitrary C_i in constant time.

Definition 13. We say that index i is *good* if: (i) $a_i = x_i = 1$ (i.e., $X(i) = 1$) and (ii) for all $j \in S_i$ (S_i is given as input), there is a pair $\langle j, \ell \rangle \in T[i]$ such that $C_j[\ell] = i$.

Let G be the set of good indices. We have the following simple observation about G.

Observation 14. There is a membership oracle verifier $\mathcal{V}_{\text{MEM}}$, which for every input i, tests if $i \in G$ in $O(t)$ time.

We argue below that the good indices make up a feasible solution. Let $\hat{x}_i = 1$ for all $i \in G$ and $\hat{x}_i = 0$ for all other i.

Lemma 15. $(\hat{x}_1, \dots, \hat{x}_n)$ *is a feasible solution of value at least* $|G|$.

Proof. The constraint enforcement structure ensures $\forall j \in [m], \sum_{i \in G} b_{ij} x_i \leqslant c_j$. Thus, $(\hat{x}_1, \dots, \hat{x}_n)$ represents a feasible solution. Note that when i is good, $\hat{x}_i = x_i = 1$. □

Using these,

Theorem 16. *There is a (t/ϵ)-time, ϵ-approximate lower bound PCP for unweighted fractional packing problems.*

Proof. We use the proof and verifier pair $\Pi_{\text{SZ}}, \mathcal{V}_{\text{SZ}}$ (from Section 3.3) to check that the size of G is at least $(1 - \epsilon)\text{OPT}$.

From our earlier observation, membership in G can be determined in time $O(t)$. Thus, the total runtime of the verifier is $O(t/\epsilon)$. □

4.2.2. Constraint enforcement: weighted version

To handle the weighted case, we modify the previous approximate PCP in two ways. First, we modify the notion of "good", so that it is still the case that a solution $\hat{x}_1, \dots, \hat{x}_n$ to the fractional packing problem that sets $\hat{x}_i$ to x_i when i is good and 0 otherwise satisfies all constraints and has value $\sum_{x_i \in G} a_i x_i$. Second, we use the approximate lower bound PCP for sums from Section 3.4 so that the verifier can guarantee that $\sum_i a_i \hat{x}_i \geqslant (1 - \epsilon)\text{OPT}$.

Since the values of the x_i's and the multipliers b_{ik} are no longer constrained to be 0/1, we need to keep track of the "space" taken up by each nonzero $b_{ij} x_i$ in each constraint. A first idea would be for the proof to contain the name of the ith variable in $b_{ij} x_i$ consecutive locations in the constraint array C_j. However, testing that a variable was allocated enough space in a constraint array would then take $O(b_{ij} x_i)$ time. Since the "resources" allocated to each variable within a constraint can be very different, we essentially keep track of the range of space taken by each variable in each constraint.

More specifically, the proof Π_{WCES} is the following *weighted constraint enforcement structure* that consists of two parts:

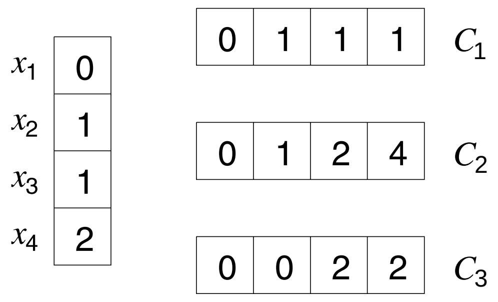

Fig. 2. The weighted case.

(1) An array X of length n such that the ith entry is the value of x_i.
(2) For the jth constraint of the form $\sum_i b_{ij}x_i \leqslant c_j$, an array of length n, where the ith entry records the running total of space taken up by the first i variables (we imagine the constraint to be a physical space of size c_j): the array C_j is $[r_1, r_2, \ldots, r_n]$ where $r_i = \sum_{k=1}^{i} b_{kj} \cdot x_k$ represents the space taken up by the first i objects, $r_i - r_{i-1}$ represents the space taken up by object i (and should be $b_{ij}x_i$), r_0 is assumed to be 0, and r_n should be at most c_j. Note that since the b_{ij}'s and x_i's are nonnegative, if the r_j's are given correctly, then they will form a nondecreasing sequence.

Note that no analogue of the third part of the proof, namely the array T, from the unweighted case is required, since here the space usage of each variable will be described in each constraint, whether or not the variable appears in the constraint. Thus, the space usage of variable i in constraint j is contained in the ith location of array C_j.

Fig. 2 shows the approximate PCP for weighted constraint enforcement used for the following problem: Maximize $x_1 + 2x_2 + 3x_3 + x_4$ subject to constraint 0: $x_1, x_2, x_3, x_4 \geqslant 0$, constraint 1: $x_1 + x_2 \leqslant 2$, constraint 2: $x_2 + x_3 + x_4 \leqslant 4$, and constraint 3: $x_1 + 2x_3 \leqslant 2$. The solution setting $x_1 = 0$, $x_2 = x_3 = 1$, $x_4 = 2$ has value 7.

In order to ensure that each constraint is satisfied, we need a definition of a "good" element that is strong enough so that the sum of all good elements does not violate any constraints and the verifier can efficiently determine whether an element is good (in particular, the verifier should not have to look at many variables in the constraint).

The proof could try to cheat by presenting a list of r_i's that is not monotone. However, if $r_{i_1-1} \leqslant r_{i_1} \leqslant r_{i_2-1} \leqslant r_{i_2} \leqslant \cdots$ form a monotone nondecreasing subsequence, then it is easy to see that the overall space taken by objects $i_1, i_2, \ldots$ together will not violate the capacity constraint, as long as all the r_i's are less than the capacity constraint. Our new definition of good borrows from the sorting spot-checker in [14].

For the purposes of the following definition, define $\succ$ to be such that $r_i \succ r_j$ if and only if $(r_i > r_j)$ OR ($r_i = r_j$ AND $i > j$), and $\prec$ analogously. We use these modifications of the $>, <$ operators in order to break ties when the elements of a list are not distinct.

Definition 17 (*Heavy element*)**.** An index i in a list $r_1, \ldots, r_n$ is said to be *heavy* if a binary search (according to the ordering relation $\succ, \prec$) for r_i is successful, i.e., finds the value r_i in location i, and finds no inconsistencies to $\succ, \prec$ along the search path.

Note that in a monotone non-decreasing list, all elements are heavy. The usefulness of the definition comes from the following fact from [14]:

Lemma 18 (see [14]). *For a pair of heavy elements r_i and r_j such that $i < j$, it must be the case that $r_i \leqslant r_j$.*

The above can be seen by noting that if k is the index of the least common ancestor of indices i and j, then since the binary searches for r_i and r_j diverge at that point, it must be that $r_i \prec r_k \prec r_j$, and so $r_i \leqslant r_j$. Thus, the heavy elements in a list form a non-decreasing subsequence. Note from the definition of a heavy element that one can test the heaviness of an arbitrary element r_i in $O(\log n)$ time.

We now define the notion of a good element as one that is represented truthfully in the constraints without violating them. Let $r_0 = 0$, $r_i = C_j[i]$, and $r_{i-1} = C_j[i-1]$.

Definition 19. We say an object i is *good* if for all $j \in S_i$: (i) $r_i - r_{i-1} = b_{ij}x_i$, (ii) $0 < r_{i-1} < r_i \leqslant c_j$, and (iii) r_i and the preceding element r_{i-1} are both heavy with respect to the list $r_1, \ldots, r_n$.

Note that, for a particular j, (i) and (ii) can be checked in $O(1)$ time, and (iii) can be checked in $O(\log n)$ time. Thus, we make the following observation.

Observation 20. There is a membership oracle verifier $\mathcal{V}_{\text{MEM}}$, which for every input i, tests if i is good in $O(t \log n)$ time.

If both the corresponding r_i and r_{i-1} are heavy for each good element in a constraint, $r_0 \geqslant 0$ and r_i is less than the capacity of the constraint, then the data structure for that particular constraint does indeed allocate space uniquely for that particular element. As a result, the sum of the good elements does not violate the constraint. Define $\hat{x}_i$ as above, by setting $\hat{x}_i$ to x_i if i is good and to 0 otherwise.

Lemma 21. *$(\hat{x}_1, \ldots, \hat{x}_n)$ is a feasible solution of value $\sum_{x_i \in G} a_i x_i$.*

Proof. Let $G = \{\ell_1, \ell_2, \ldots, \ell_k\}$ be the good indices (assume $\ell_1 < \ell_2 < \cdots < \ell_k$). Consider the jth constraint. Then, since the heavy elements form a monotone non-decreasing subsequence and are all at most c_j, $\sum_i b_{ij}\hat{x}_i = \sum_{i \in G} b_{ij}\hat{x}_i = \sum_{i \in G}(r_i - r_{i-1}) \leqslant r_{\ell_k} \leqslant c_j$. □

Theorem 22. *There is a $((t/\epsilon)\log n)$-time, ϵ-approximate lower bound PCP for fractional packing problems.*

Proof. We use the proof and verifier for lower bounding sums from Section 3.4 to check that the weight of the good elements (i.e., $\sum_{i \in G} b_{ij}x_i$) is at least $(1-\epsilon)$OPT . Using Observation 20, the theorem follows. The total runtime of the verifier is $O((t/\epsilon)\log n)$. □

4.2.3. Other applications of constraint enforcement

The constraint enforcement structure can be applied to several optimization problems. We give a few examples to demonstrate the scope of the technique.

Maximum flow. A graph G with capacity constraints on the edges and special nodes s, t is given. If there is a flow of size f, then it can be proved to the verifier $\mathcal{V}_{\text{FLOW}}$ that a flow of size $\geqslant (1-\epsilon)f$ exists by the following method. Note that to verify that a flow is legal, the verifier must verify that the solution observes conservation of flow at each node and capacity constraints at each edge. The proof Π_{FLOW} consists of a list T of path-flows that combine to make up the flow of size f. The verifier $\mathcal{V}_{\text{FLOW}}$ picks random path-flows and ensures that they are "good" by checking that the flow is correctly packed into each edge that it follows—in doing so, it ensures the path-flow satisfies conservation of flow at each node along the path from s to t. Since each path-flow is of length at most n (the number of vertices), we have an n-sparse packing problem. The constraint enforcement structure ensures that no more than c_{uv} capacity is needed to accommodate all of the path flows simultaneously on each edge (u, v). For relatively small flows, we use the fact that any flow of integer magnitude f can be decomposed into f unit size path-flows. The approximate PCP for the unweighted version of constraint enforcement can be used inside the proof Π_{FLOW} to give a proof by which the verifier can determine that there are enough good unit path flows in time $O(n/\epsilon)$. From the above, it is easy to see:

Theorem 23. *There is an $((n/\epsilon))$-time, $|f|$-space, ϵ-approximate lower bound PCP for the maximum flow problem.*

For larger flows, it may be desirable to find a proof whose size is polynomial in n, even at the cost of requiring a slightly less efficient verifier. We use the result of Edmonds and Karp [13] that shows that any flow can be decomposed into at most m (where m is the number of edges in the graph) path-flows. The approximate PCP for weighted constraint enforcement can be used to give a verifier with runtime $O((n/\epsilon)\log n)$. From the above, it is easy to see:

Theorem 24. *There is an $((n/\epsilon)\log n)$-time, ϵ-approximate lower bound PCP for the maximum flow problem in which the proof size is* $\text{poly}(n)$.

The constraint enforcement structure can also be used to show a lower bound on the size of a multi-commodity flow, in which the runtime of the verifier is $O((qn/\epsilon)\log n)$, where q is the number of commodities.

Bin packing. A set of n weighted objects, a bin size B, and an $\epsilon < 1$ are given. If it is possible to pack the objects into p bins, then there is a proof that convinces the verifier that $p + \epsilon n$ bins are sufficient: the proof will use the constraint enforcement structure to assure the verifier that at least $(1-\epsilon)$ fraction of the objects can be packed into p bins. The bound follows by noting that the other objects, if they do not already fit into the p bins, can each be placed into their own bin. The running time of the verifier is $O((1/\epsilon)\log n)$. From the above, it is easy to see:

Theorem 25. *There is an (ϵn)-additive approximate upper bound PCP for the bin packing problem.*

Exact cover by k-sets, matching. Given set X with $|X| = kq$ and a collection C of k-element subsets of X, does C contain an exact cover for X, i.e., a sub-collection $E \subseteq C$ such that every element of X occurs in exactly one member of E? We consider a maximization version of this problem—to maximize the number of elements in X that are covered uniquely. We argue that there is a proof to convince the verifier that there exists a partial covering F that covers at least $1-\epsilon$ fraction of the elements of X such that no element in X is covered by more than one set. The proof utilizes the unweighted constraint enforcement structure: For each set $s_i \in C$ there is a variable x_i that is set to 1 if $s_i \in F$ and 0 otherwise. For each element in X there is a constraint which ensures that it is contained in at most one of the sets in F: b_{ij} is 1 if set s_i contains element j. For each $c \in F$ such that $c = \{a_1, \ldots, a_k\}$, c should appear in $C_{a_1}, \ldots, C_{a_k}$. If the verifier samples the $c \in F$ and decides that most are good, then it can conclude that there is a collection $F' \subseteq F$ such that $|F'| \geqslant (1-\epsilon)|F|$ and such that no $a \in X$ is covered more than once by F'. This gives us the following theorem.

Theorem 26. *There is a (k/ϵ)-time, ϵ-approximate lower bound PCP for the exact cover by k-sets problem.*

Note that, since a matching is a cover by two-sets, the method can be used to show an approximate lower bound PCP on the size of a matching in a graph.

Shop scheduling. In the *open shop scheduling* problem, a set of p products, m work teams, and a deadline D are given. Each product consists of m tasks, each designated to be processed by a different work team j at some point during production. Task j of product x_i takes t_{ij} time units to complete. A product can be with at most one team, and a team can be working on at most one product at any given time. If it is possible to complete all p products before deadline D, then there is a proof that can convince a $O((m/\epsilon)\log p)$-time verifier that at least $(1-\epsilon)p$ products can be completed before the deadline. The proof uses the weighted constraint enforcement structure to ensure that products are with at most one team and that teams are working on at most one product at any given time. Variants of the above problem, such as flow shop and job shop scheduling can be handled in a similar manner.

Subset sum. Given $x_1, \ldots, x_n$ and a bound B, using the protocol for lower bound on sums (Section 3.4) as well as the weighted constraint enforcement structure, there is a proof to convince an $O((1/\epsilon)\log n)$-time verifier that there exists a set S such that $B(1-\epsilon) \leqslant \sum_{i \in S} x_i \leqslant B$. A similar result holds for partition.

4.3. Matching problems

In this section we consider problems based on matching. We first given an alternate approximate PCP for matching that does not use constraint enforcement structure and then consider the problem of minimum maximal matching.

Matching. The following approximate PCP can be used to show a lower bound on the size of a matching of a graph $G = (V, E)$. In particular, the proof can convince the verifier that G has a matching of size at least $k(1-\epsilon)$ by presenting a purported matching L of size k. The proof Π_{MAT} consists of three parts.

(1) A list L of edges in the matching.
(2) A proof that $|L \cap E| \geqslant (1-\epsilon/2)k$. This can be accomplished using the proof Π_{sz} for lower bounding set size (Section 3.3).
(3) A proof that at most $\epsilon/2$ fraction of edges involve vertices that are matched more than once. This can be accomplished via an array T such that for each vertex $v \in V$, $T[v]$ points to its

matched edge (if v is matched at all) in L, i.e., $T[v] = i$ if $L[i] = (v, *)$ or $L[i] = (*, v)$ and $T[v] = 0$ otherwise.

The verifier $\mathcal{V}_{\text{MAT}}$ can check (2) using the verifier $\mathcal{V}_{\text{SZ}}$ and can check (3) by choosing a random edge $L[i] = (u, v)$ from L, then choosing a random vertex $w \in_R \{u, v\}$ and checking if $\Pi.T[w] = i$. From the above, it is easy to see:

Theorem 27. *There is a $(1/\epsilon)$-time, ϵ-approximate lower bound PCP for matching.*

Minimum maximal matching. Given a graph $G = (V, E)$, $|V| = n$, of degree at most d, does G contain a maximal matching of size at most U? This problem is NP-complete if $d \geqslant 3$. If there is such a matching, then there is a proof that can convince a verifier (in $O(1/\epsilon)$ time) that there is a maximal matching of size at most $U + n\epsilon/2$. The proof Π_{MMAT} consists of two parts.

(1) An array L of size U that contains the edges of the matching.
(2) An array T such that for each vertex $v \in V$, $T[v]$ points to its matched edge (if v is matched at all) in L, i.e., $T[v] = i$ if $L[i] = (v, *)$ or $L[i] = (*, v)$ and $T[v] = 0$ otherwise.

To check this proof, the verifier $\mathcal{V}_{\text{MMAT}}$ picks a random node u and if u is unmatched (i.e., $\Pi.T[u] = 0$), then makes sure (using $\Pi.T$) the neighbors of u are also unmatched. If the number of unmatched nodes with unmatched neighbors is more than ϵn, the verifier is likely to output FAIL. The bound follows since there exists a pairing of unmatched nodes such that one needs to add at most one edge for every pair. From the above, it is easy to see:

Theorem 28. *There is an $(\epsilon n/2)$-additive approximate upper bound PCP for minimum maximal matching.*

Acknowledgment

We thank the reviewers for their help in improving the presentation of this paper.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, Proof verification and hardness of approximation problems, Journal of the ACM 45 (3) (1998) 501–555.
[2] S. Arora, S. Safra, Probabilistic checkable proofs: a new characterization of NP, Journal of the ACM 45 (1) (1998) 70–122.
[3] L. Babai, L. Fortnow, L. Levin, M. Szegedy, Checking computations in polylogarithmic time, in: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, 1991 pp. 21–31.
[4] M. Ben-Or, S. Goldwasser, J. Kilian, A. Wigderson, Multi-prover interactive proofs: How to remove intractability assumptions, in: Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 1988 pp. 113–131.
[5] M. Blum, S. Kannan, Designing programs that check their work, Journal of the ACM 42 (1) (1995) 269–291.
[6] M. Blum, M. Luby, R. Rubinfeld, Self-testing/correcting with applications to numerical problems, Journal of Computing and System Sciences 47 (3) (1993) 549–595, Science 1592 (1999) 402–414.
[7] J.Y. Cai, R. Lipton, R. Sedgewick, A. Yao, Towards uncheatable benchmarks, in: Proceedings of the 8th Structure in Complexity Theory, 1993, pp. 2–11.
[8] R. Canetti, G. Even, O. Goldreich, Lower bounds for sampling algorithms for estimating the average, IPL 53 (1) (1995) 17–25.

[9] H. Chernoff, A measure of the asymptotic efficiency for tests of a hypothesis based on the sum of observations, Annals of Mathematical Statistics 23 (1952) 493–509.

[10] A. Condon, Space bounded probabilistic game automata, Journal of the ACM 38 (2) (1991) 472–494.

[11] P. Dagum, R. Karp, M. Luby, S. Ross, An optimal algorithm for Monte Carlo estimation, SIAM Journal on Computing 29 (5) (2000) 1484–1496.

[12] C. Dwork, L. Stockmeyer, Finite state verifiers I: The power of interaction, Journal of the ACM 39 (4) (1992) 800–828.

[13] J. Edmonds, R. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, Journal of the ACM 19 (2) (1972) 248–264.

[14] F. Ergun, S. Kannan, R. Kumar, R. Rubinfeld, M. Viswanathan, Spot-checkers, Journal of Computing and System Sciences 60 (3) (2000) 717–751.

[15] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, M. Szegedy, Interactive proofs and the hardness of approximating cliques, Journal of the ACM 43 (2) (1996) 268–292.

[16] E. Fischer, On the strength of comparisons in property testing, Technical Report, Electronic Colloquium on Computational Complexity, TR01-008, 2001.

[17] L. Fortnow, The complexity of perfect zero-knowledge, Randomness and Computation 5 (1989) 327–343.

[18] L. Fortnow, C. Lund, Interactive proof systems and alternating time–space complexity, Theoretical Computer Science 113 (1993) 55–73.

[19] L. Fortnow, J. Rompel, M. Sipser, On the power of multi-prover interactive protocols, Theoretical Computer Science 134 (2) (1994) 545–557.

[20] L. Fortnow, M. Sipser, Interactive proof systems with a log space verifier, in: L. Fortnow (Ed.), Complexity-Theoretic Aspects of Interactive Proof Systems, Ph.D. Thesis, MIT, 1989.

[21] M. Furer, O. Goldreich, Y. Mansour, M. Sipser, S. Zachos, On completeness and soundness in interactive proof systems, Advances in Computing Research: A Research Annual, Randomness and Computation 5 (1989) 429–442.

[22] P. Gemmell, R. Lipton. R. Rubinfeld, M. Sudan, A. Wigderson, Self-testing/correcting for polynomials and for approximate functions, in: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, 1991, pp. 32–42.

[23] O. Goldreich, S. Goldwasser, D. Ron, Property testing and its connection to learning and approximation, Journal of the ACM 45 (4) (1998) 653–750.

[24] O. Goldreich, S. Micali, A. Wigderson, Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems, Journal of the ACM 38 (1) (1991) 691–729.

[25] O. Goldreich, D. Ron, Property testing in bounded degree graphs, Algorithmica 32 (2) (2002) 302–343.

[26] O. Goldreich, D. Ron, A sublinear bipartite tester for bounded degree graphs, Combinatorica 19 (3) (1999) 335–373.

[27] S. Goldwasser, M. Sipser, Private coins versus public coins in interactive proof systems, in: Proceedings of the 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 59–68.

[28] S. Khanna, M. Sudan, L. Trevisan, Constraint satisfaction: the approximability of minimization problems, in: Proceedings of the 12th IEEE Conference on Computational Complexity, 1997, pp. 282–296.

[29] J. Kilian, Zero-knowledge with logspace verifiers, in: Proceedings of the 29th Annual Symposium on Foundations of Computer Science, 1988, pp. 25–35.

[30] J. Kilian, A note on efficient zero-knowledge proofs and arguments, in: Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, pp. 723–732.

[31] J. Kilian, Improved efficient arguments (preliminary version), in: Proceedings of the Advances in Cryptology—CRYPTO, Springer Lecture Notes in Computer Science, vol. 963, 1995, pp. 311–324.

[32] R. Lipton, New directions in testing, in: Proceedings of the DIMACS Workshop on Distr. Comp. and Cryptography, 1991, pp. 191–202.

[33] R.C. Merkle, A certified digital signature, in: Proceedings of the Advances in Cryptology—CRYPTO, Springer Lecture Notes in Computer Science, vol. 435, 1989, pp. 218–238.

[34] S. Micali, CS proofs, in: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 436–453.

[35] S. Plotkin, D. Shmoys, E. Tardos, Fast approximation algorithms for fractional packing and covering problems, in: Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, 1991, pp. 495–504.

[36] A. Polishchuk, D. Spielman, Nearly-linear size holographic proofs, in: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, 1994, pp. 194–203.

[37] R. Rubinfeld, M. Sudan, Robust characterizations of polynomials and their applications to program testing, SIAM Journal on Computing 25 (2) (1996) 252–271.
[38] T. Schaefer, The complexity of satisfiability problems, in: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 216–226.
[39] D.A. Spielman, Linear-time encodable and decodable error-correcting codes, IEEE Transactions on Information Theory 42 (6) (1996) 1723–1732.
[40] M. Szegedy, Many-valued logics and holographic proofs, in: Proceedings of the 26th International Colloquium on Automata, Languages, and Programming, Springer Lecture Notes in Computer Science, vol. 1644, 1999, pp. 676–686.