Fast approximate PCPs *

Funda Ergün[†] Ravi Kumar[‡] Ronitt Rubinfeld[§]

Abstract

We investigate the problem of when a prover can aid a verifier to reliably compute a function *faster* than if the verifier were to compute the function on its own. We focus on the case when it is enough for the verifier to know that the answer is *close* to correct. We use a model of proof systems which is based on interactive proof systems, probabilistically checkable proof systems, program checkers, and CS proofs. We develop protocols for several optimization problems, in which the running time of the verifier is significantly less than the size of the input. For example, we give polylogarithmic time protocols for showing the existence of a large cut, a large matching and a small bin packing. In contrast, the protocols used to show that IP $=$ PSPACE, MIP $=$ NEXP and NP $=$ PCP$(\lg n, 1)$ [Sha90, BFL91, ALM+98, BFLS90] require a verifier that runs in $\Omega(n)$ time. In the process, we develop a set of tools for use in constructing these proof systems.

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 a prover 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? In this paper, we consider the case when it is enough for the client to know that the answer is *close* to correct. We show that for a number of interesting functions, a very short dialogue between the client (the verifier) and the company (the prover) can convince the client of the approximate correctness of the answer. In particular, we give examples in which the verifier can be convinced in time sublinear in the input size.

This scenario motivates an investigation of highly efficient proof systems for *specific* computational problems, even when the problems are known to be solvable in polynomial time. Our focus is on finding the best implementations of such proof systems in terms of the running time of the verifier. It is known that IP $=$ PSPACE, MIP $=$ NEXP, NP $=$ PCP$(\lg n, 1)$ [LFKN90, Sha90, BFL91, ALM+98]. From the work of [BFLS90] and [Spi96], it is possible to construct proof systems for any proof in a reasonable formal system with an $O(n + \lg \ell)$-time verifier, where n is the length of the theorem and ℓ is the length of the proof. 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 communication between the prover and verifier is limited. Note that the protocols in the aforementioned results all require that the verifier look at the whole input, and thus do not give sublinear time protocols.

The model we consider, described in Section 2, is based on the models of IP [GMR89], PCP [FRS94], and CS proofs [Mic94], with modifications borrowed from the models of program checking [BK95], approximate program checking [GLR+91], property testing [RS96, GGR98] and spot-checking [EKK+98].

OUR RESULTS. 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

* This work was partially supported by ONR N00014-97-1-0505, MURI, NSF Career grant CCR-9624552, and an Alfred P. Sloan Research Award.

† Bell Laboratories, 700 Mountain Avenue, Murray Hill, NJ 07974. email: ergun@research.bell-labs.com

‡ IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120. email: ravi@almaden.ibm.com

§ Department of Computer Science, Cornell University, Ithaca, NY 14850. email: ronitt@cs.cornell.edu. Currently on sabbatical leave at IBM Almaden Research Center.

constraint satisfaction problems, including max cut and max SAT, to a polylogarithmic time verifier. We show how to prove the existence of a near optimal solution of a sparse fractional packing problem to a polylogarithmic time verifier. Our fractional packing protocol can be used for several other problems. For example, we show that 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 can be computed efficiently by the prover. In all of the above protocols it is also possible to prove the existence of suboptimal solutions, i.e., if the prover knows of a solution of value v, it can prove the existence of a solution of value at least $(1-\epsilon)v$. We next investigate methods of proving additive approximations of bounds on the sizes of unions and intersections of several sets such that the verifier requires only logarithmic time. One application of such protocols is to estimating the size of unions or intersections of several database queries. Finally, we consider spot-checking and property testing and show that more efficient results for testing bipartiteness and element distinctness can be achieved with the aid of a prover.

We develop a new set of tools for use in constructing these proof systems. For example, we give constant time protocols for estimating lower bounds on the size of a set and for estimating lower bounds on sums of n inputs. We develop a constraint enforcement protocol which allows the verifier to ensure that linear upper bound constraints are satisfied without looking at all of the variables involved. Some of our techniques rely on simplifications of the methods of [GS86, For89] by which a prover can convince a verifier that a set is of approximately a certain size.

2 The model

Interactive proof systems (IPS) [GMR89] and probabilistically checkable proof systems (PCPS) [FRS94] (equivalent in power to multiple prover proof systems [BGKW88], see also [FGL+96, AS98, BFLS90]) can be used to convince a polynomial time verifier of the correctness of a decision problem computation. Definitions of IP which parametrize the runtime of the verifier appear in [Con91, FL93]. CS proofs [Mic94] extend the model in several ways: by applying more generally to function computations as well as problems above NEXP; and by restricting the runtime of the prover.

Program result checking [BK95] and self-testing/correcting techniques [BLR93, Lip91] were introduced so that a client could ensure the correctness of a solution to a computation. Program result checkers can be viewed as a special type of proof system for function computations, in which the prover is restricted to answering other instances of the same computational problem. It turns out that all result checkers as well as result checkers in the library setting [BLR93] satisfy the requirements of the model used here.

Proving that results are approximately correct is also related to approximate checking [GLR+91], property testing [RS96, GGR98], and spot-checking [EKK+98], where the goal is to determine whether an answer is close to correct for various notions of closeness. All approximate checkers satisfy the requirement of the model here. Conversely, all of our results can be viewed as spot-checkers which use the aid of a prover.

The model we use is based on the above models and in particular: applies to function computations and decision, optimization, approximation, and search problems; allows the prover to prove the weaker assertion that a solution is approximately correct; parametrizes the run time of the verifier, for example to distinguish linear time verifiers from quadratic time verifiers; and analyzes the run time of the verifier implemented as a RAM machine in order to understand the exact asymptotic complexity of the verifier. We will not assume any bounds on the computation time of the prover.

Both the prover and verifier are interactive RAM machines that have read access to an input and an output tape, read/write access to communication tapes, read access to a public or private source of random bits, and read/write access to private computation tapes. We assume that the verifier can access any word in any tape in constant time.

We give definitions for both the approximate IP and PCP models at the same time. The only distinction between the two is that in the approximate PCP model, the prover is restricted to a function determined before the start of the interaction.

Definition 1 *Let $\Delta(\cdot,\cdot)$ be a distance function. A function f is said to have an $t(\epsilon, n)$-*approximate interactive proof (probabilistically checkable proof) system *with distance function Δ if there is a randomized verifier $\mathcal{V}$ such that for all inputs ϵ and x of size n, the following holds. Let y be the contents of the output tape, then:*

1. *If $\Delta(y, f(x)) = 0$, there is a (function-restricted) prover $\mathcal{P}$, such that $\mathcal{V}$ outputs pass with probability at least 3/4 (over the internal coin tosses of $\mathcal{V}$);*
2. *If $\Delta(y, f(x)) > \epsilon$, for all* (function-restricted) *provers $\mathcal{P}'$, $\mathcal{V}$ outputs fail with probability at least 3/4 (over the internal coin tosses of $\mathcal{V}$); and*
3. *$\mathcal{V}$ runs in $O(t(\epsilon, n))$ time.*

The interactive (probabilistically checkable) proof protocol can be repeated $O(\lg 1/\delta)$ times to get confidence $\geq 1-\delta$. We omit all dependence on δ from our protocols throughout this paper.

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 interactive proof systems for decision problems require that when $y = f(x)$, an honest prover can convince the verifier of that fact, and when $y \neq f(x)$, no prover can convince the verifier of that. In our model, this is achieved by choosing $\Delta(\cdot, \cdot)$ such that $\Delta(y, y') > \epsilon$ whenever $y \neq y'$ and $\Delta(y, y) = 0$. Note that the output of $\mathcal{V}$ is not specified when $0 < \Delta(y, f(x)) \leq \epsilon$. To give a proof system which passes only when $y > (1 - \epsilon)f(x)$, one could use the function $\Delta(y, y') = \max\{0, 1 - y/y'\}$. We refer to an IPS/PCPS with this distance function as an *approximate lower bound* IPS/PCPS.

For notational convenience, we often mix notions of interactive and probabilistically checkable proofs by using both within the same protocol, referring to a prover sending information as well as permanently writing down information before the start of the protocol (which corresponds to committing to a set of responses to queries that will be made later in the protocol). These systems can clearly be simulated by a function-restricted prover, since $\mathcal{P}$ can decide on all of its responses before the start of the protocol. Unless otherwise noted, all protocols are described for the PCP model.

INTERACTIVE SPOT-CHECKING MODEL. We give a more general definition of IPS/PCPS which applies to distance functions that correspond to property testing and spot-checking. We define an *interactive-spot-checker*, which is essentially a spot-checker [EKK+98] that is allowed the assistance of a prover.

Definition 2 *Let $\Delta(\cdot, \cdot)$ be a distance function. We say that $\mathcal{V}$ is an $t(\epsilon, n)$*-interactive-spot-checker (*ISC*) *for f with distance function Δ if, given any input x, claim y for the value of $f(x)$, and ϵ,*

1. *If $\Delta(\langle x, y\rangle, \langle x, f(x)\rangle) = 0$, then there is a function-restricted prover $\mathcal{P}$, such that $\mathcal{V}$ outputs pass with probability at least 3/4 (over the internal coin tosses of $\mathcal{V}$);*
2. *If for all inputs x', $\Delta(\langle x, y\rangle, \langle x', f(x')\rangle) > \epsilon$, then for all function-restricted provers $\mathcal{P}'$, $\mathcal{V}$ outputs fail with probability at least 3/4 (over the internal coin tosses of $\mathcal{V}$); and*
3. *$\mathcal{V}$ runs in $O(t(\epsilon, n))$ time.*

The choice of the distance function Δ is problem specific, and determines the ability to spot-check. The condition on the runtime of the spot-checker enforces the "little-oh" property of [BK95], i.e., as long as f depends on all bits of the input, the condition on the runtime of the spot-checker forces the spot-checker to run faster than any correct algorithm for f, which in turn forces the spot-checker to be different than any algorithm for f. Note also that Δ can be set to 0 for many of the inputs and Δ need not be computable by the verifier, so that this definition allows interactive proofs for promise problems.

USING PCPS OVER THE INTERNET. When interacting over the internet, the verifier may want some assurance that $\mathcal{P}$ is function-restricted, without resorting to having the prover transmit the whole proof in advance of the verification process. One possibility is to use a trusted third party: $\mathcal{P}$ transmits the proof to the third party, and the verifier interacts with the third party assuming that it has no reason to change pieces of the proof. Alternatively, one can bound the running time of $\mathcal{P}$, as is done in the model of CS proofs [Mic94]. Then it is possible to force the prover to commit to the proof in such a way that only provers that are computationally more powerful than the allowed bound are able to change the proof in a convincing way. One can use commitment methods [Mer90] in this setting [Kil92, Mic94, Kil94]. Recently, a third method was proposed by [CMS99] who show that techniques from *private information retrieval* (PIR) [CGKS95] can be used to force a prover to commit to a proof. In this latter work, the essential idea is that since the prover does not know which bit of the proof is being revealed to the verifier, the prover cannot change the proof in a convincing manner. Note that PIR schemes with computationally bounded provers are more efficient (cf. [CGKS95, KO97, CMS99]).

RELATED MODELS. Several other works have looked at IPS/PCPS with resource limited verifiers, especially verifiers using logarithmic space. In [Con91, FL93, DS92, FS88], the question of classifying the languages that have interactive proofs with various models of space-bounded verifiers is studied. The work of [DS92, Kil] consider the issue of when zero-knowledge interactive proof systems exist for systems with space bounded verifiers. Related work of [CLSY90] 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 does not care how the prover computed the answer, only that the answer is correct.

NOTATION. We use $x \in_R S$ to denote that x is chosen uniformly at random from S. We assume the size of a word is b bits and we assume all integer variables fit in a word.

3 Some basic building blocks

3.1 Permutation enforcement (multiset equality)

Given an input list $X = \langle x_1, \ldots, x_n\rangle$, many of our protocols require that the prover rewrite the list in a different order $Y = \langle y_1, \ldots, y_n\rangle$ (for example, the sorted order). We would like the verifier to be able to ensure that $|X \cap Y| \geq (1 - \epsilon)n$. In particular, the verifier should be able to access elements from Y while ensuring that each accessed element corresponds to a single location in X. The difficulty comes from the possibility that neither list is necessarily distinct. One would like to prevent the possibility that an x_i from X was duplicated 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 the aid of a prover, $\mathcal{V}$ requires $\tilde{\Theta}(\sqrt{n})$ time to ensure that $|X \cap Y| \geq (1-\epsilon)n$ [EKK+98]. Here we show that it can be done in $O(1/\epsilon)$ time.

The *permutation enforcer* consists of 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. Similarly, the contents of location i in T_2 contains a pointer to the location of y_i in X.

Let i be *good* if $T_1[T_2[i]] = i$ and $x_i = y_{T_1[i]}$. Then it is easy to see that:

Lemma 3 $|X \cap Y| \geq |\{i \mid i \text{ is good}\}|$.

Thus, to verify that $|X \cap Y| \geq (1-\epsilon)n$, the verifier should choose $O(1/\epsilon)$ random i's and output fail if it ever finds an i that is not good. If $X = Y$, the permutation enforcer will always cause $\mathcal{V}$ to pass, and if $|X \cap Y| \leq (1-\epsilon)n$, no matter what $\mathcal{P}$ writes in place of the permutation enforcer, $\mathcal{V}$ will fail with probability at least $3/4$.

Let $f(X, Y) = 1$ if $X = Y$ and 0 otherwise. Given two multisets X, Y, let $\rho(X, Y)$ be the minimum number of elements that need to be inserted to or deleted from X in order to obtain Y. Then $\Delta(\langle (X,Y), Z\rangle, \langle (X', Y'), f(X', Y')\rangle)$ is infinite if either $X \neq X'$ or $Z \neq f(X', Y')$, and otherwise is $\rho(Y, Y')/|Y|$. One can see that this definition of Δ ensures $\mathcal{V}$ passes only multisets X and Y that are at least close to equal.

Theorem 4 *Given two multisets of size n and constant ϵ, there is an $(1/\epsilon)$-ISC for multiset equality with distance function Δ.*

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 $O(1/\epsilon)$ time protocol by which the verifier can ensure that the number of distinct elements in X is at least $(1-\epsilon)n$. Without the aid of the prover, $\mathcal{V}$ requires $\Omega(\sqrt{n})$ time to determine the same [EKK+98]. The protocol we use is a simplification of a protocol given by [For89] in order to allow a prover to convince a verifier of an upper bound on the size of a set. Interestingly, we use the same technique to give a lower bound on the size of a set.

```
Repeat O(1/ε) times:
  V chooses i ∈_R [1...n]
  V sends x_i to P
  P returns j to V
  V fails if i ≠ j
```

If X is distinct, then $\mathcal{P}$ can answer so that $\mathcal{V}$ always passes. If the number of distinct elements in X is less than $(1-\epsilon)n$, then for all provers $\mathcal{P}'$, $\mathcal{V}$ fails with probability at least $3/4$. More formally, let $f(X) = 1$ if X is distinct and 0 otherwise. Define $\Delta(\langle X, Y\rangle, \langle X', f(X')\rangle)$ to be infinite if $Y \neq f(X')$, and $\rho(X, X')/|X|$ otherwise (ρ is as defined previously). Note that it is important for the correctness of the protocol that $\mathcal{P}$ is restricted to a function determined before the start of the interaction.

Theorem 5 *Given a multiset of size n and constant ϵ, there is an $(1/\epsilon)$-ISC for element distinctness with distance function Δ.*

Proof: If the multiset X is distinct, $\mathcal{P}$ can always find $j = i$. If the number of distinct elements in X is less than $(1-\epsilon)n$, the probability that $\mathcal{V}$ chooses an i corresponding to a nondistinct 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, $\mathcal{V}$ will fail with probability at least $3/4$. □

A SPACE EFFICIENT PROOF. If the function-restricted $\mathcal{P}$ in the previous protocol is implemented by having $\mathcal{P}$ write down the answers to all queries of $\mathcal{P}$ in advance of the conversation, $\mathcal{P}$ writes a table of size proportional to a bound on the maximum value of x_i. It is possible to save space, by using an algorithm in which $\mathcal{V}$ runs in $O((1/\epsilon) \cdot \lg n)$ time: $\mathcal{P}$ writes a list of 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. $\mathcal{V}$ then performs binary search to find (x_i, j) and checks $j = i$.

3.3 Proving lower bounds on the size of a set

Given a set S represented by a list enumerating its elements, it is nontrivial to deduce the size of S from the size of the list, since it is not known whether the elements in the list are distinct. Given a method by which $\mathcal{V}$ can determine whether a b-bit element x is in S (for example, if S is in fact represented by a list, $\mathcal{V}$ could be convinced in constant time that $x \in S$ if $\mathcal{P}$ sends $\mathcal{V}$ a pointer to the location of x in S), $\mathcal{V}$ could estimate $|S|/2^b$ to within a multiplicative error of ϵ by sampling. This requires $\Omega(2^b/(\epsilon|S|))$ samples [DKLR95, CEG95]. The two methods described here are more efficient, where the running times are described in terms of γ, an upper bound on a IP (or a PCP) protocol by which $\mathcal{P}$ can convince $\mathcal{V}$ that $x \in S$. The first protocol is faster, but the second protocol (due to [GS86]) can be performed directly in an IP setting.

The following protocols allow $\mathcal{P}$ to convince $\mathcal{V}$ that the size of S is at least $(1-\epsilon)|S|$ for any $\epsilon > 0$. In particular, let p be $\mathcal{P}$'s claimed size of S, then if $|S| > p$ the protocol always passes and if $|S| \leq (1-\epsilon)p$ the protocol fails with probability at least $3/4$.

FAST LOWER BOUNDS ON SET SIZES IN THE PCP MODEL. The following protocol uses the protocols of the previous sections such that each has probability of error at most 1/8. An auxiliary array A will be used to refer to both an array used to represent the set and the multiset which is defined by its contents.

```
𝒫 sends p to 𝒱
𝒫 writes the elements of S
  to an array A of size p
Perform element distinctness protocol
  on A with parameter ε/2
Repeat O(1/ε) times:
  𝒱 sends i ∈_R [1,...,p] to 𝒫
  𝒫 sends 𝒱 a proof A[i] ∈ S
```

Clearly if $|S| > p$, there $\mathcal{V}$ will pass. Conversely, $\mathcal{V}$ ensures that the fraction of distinct elements in A is at least $(1-\epsilon/2)$ and that at most $\epsilon/2$ fraction of the elements are not in S. Thus, $|A \cap S| \geq (1-\epsilon)p$.

Theorem 6 *There is an (γ/ϵ)-approximate lower bound* PCP *for the size of a set.*

LOWER BOUNDS ON SET SIZES IN THE IP MODEL. The following protocol is due to [GS86]. Let $c \geq \lg 0.2/\lg(1-\epsilon)$. Let n be such that $2^{n-1} \leq p^c \leq 2^n$. Let $\mathcal{H}$ be a set of pairwise independent hash functions mapping cb bit strings to n bit strings.

```
𝒫 sends p to 𝒱
Repeat the following O(1) times:
  𝒱 sends 𝒫, y ∈_R [0...2^n − 1] and h ∈_R ℋ
  𝒫 returns x_1,...,x_c ∈ S^c, along with
    proofs of their containment in S
    such that h(x_1 ∘ ... ∘ x_c) = y
  𝒱 checks that the proofs are valid
𝒱 passes if all proofs are valid for
  at least 9/40 of repetitions
```

Theorem 7 *There is an $((\frac{-1}{\lg 1-\epsilon} \cdot (\gamma + b) + t_1(b,n) + t_2(b,n)))$-approximate lower bound* IP *for the size of a set, where t_1 is the time required to evaluate a hash function $h \in \mathcal{H}$, and t_2 is the time required to generate a random $h \in \mathcal{H}$.*

The idea of the proof is that if S is as big as claimed, then $\mathcal{P}$ will be able to find $x_1, \ldots, x_c$ with probability at least $1/4$. If $S \leq (1-\epsilon)p$, then the success probability of $\mathcal{P}$ will be at most $1/5$.

3.4 Approximate lower bounds on sums

Given positive integers $x_1, \ldots, x_n$, we show how $\mathcal{P}$ can convince $\mathcal{V}$ of a good approximation to a lower bound on $\sum_{i=1}^n x_i$. Without aid of a prover, $\mathcal{V}$ 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 prover can convince the verifier that the sum is at least $1-\epsilon$ times the claimed value. The first requires only that $\mathcal{V}$ use constant time but requires a very large proof size. The latter requires that $\mathcal{V}$ spend $O(\lg B)$ time, where B is an upper bound on the x_i's (since we assume x_i fits in a word, $B < 2^b$).

USING LOWER BOUND PROTOCOLS. Consider the set $S = \{(i,j) \mid 1 \leq i \leq n,\ 1 \leq j \leq x_i\}$ (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 given (i,j), $\mathcal{V}$ can determine membership in S in constant time: first verify that $1 \leq i \leq n$ and then that $1 \leq j \leq x_i$. The lower bound protocols of the previous section may be used to estimate $|S|$ such that $\mathcal{V}$'s running time is $O(\frac{1}{\epsilon})$. If the weights are too large to fit in a word, then it is enough to work with only the $O(1/\epsilon)$ most significant bits of the weights.

Theorem 8 *There is an $(1/\epsilon)$-approximate lower bound* PCP *for the sum of n integers.*

GROUPING ELEMENTS BY SIZE. In the second protocol, $\mathcal{V}$ uses random sampling to estimate the sum. Since the number of samples required to get good estimates depends on the variance of the sample, $\mathcal{P}$ helps $\mathcal{V}$ by putting the x_i's in groups for which the variance is small: $\mathcal{P}$ groups the x_i's such that the i-th group contains all x_i whose weights are between $B/2^i$ and $B/2^{i+1}$. Since we assume integer weights, there are at most $\lg B$ such groups. However, $\mathcal{V}$ must ensure, using the permutation enforcer, that in fact each element sampled comes from the original set of x_i's. Suppose $\mathcal{V}$ chooses element y_j in one of the groups. Say that y_j is *good* if the permutation enforcer is consistent, i.e., $T_1[T_2[j]] = j$ and $x_i = y_{T_1[i]}$ and y_j's weight is consistent with its group. Let $G = \{j \mid y_j \text{ is good}\}$. Then $\sum_{j\in G} y_j \leq \sum_i x_i$. $\mathcal{V}$ uses sampling to lower bound $\sum_{j \in G} y_j$. To do this, suppose the i-th group has n_i elements. Then $\mathcal{V}$ picks $O(1/\epsilon)$ elements, checks that they are good, and sets S_i to be their average multiplied by n_i. This gives an ϵ-approximation for a lower bound on the sum of the elements in the i-th group (cf. [DKLR95]). Finally, $\mathcal{V}$ outputs $\sum S_i$, the total sum, which lower bounds $\sum x_i$ to within a multiplicative factor of $1-\epsilon$. The total running time is $O(\frac{1}{\epsilon}\lg B)$.

Theorem 9 *There is an $((1/\epsilon)\lg B)$-approximate lower bound* PCP *for the sum of n integers, where the proof size is $O(n)$.*

4 PCPS for optimization and graph problems

4.1 Constraint satisfaction problems

A WARMUP: LOWER BOUNDS ON THE CUT SIZE. We give a protocol by which $\mathcal{V}$ can be quickly convinced by $\mathcal{P}$ that a given graph $G = (V,E)$ has a large cut. In this warmup case, the PCP is especially straightforward. The main idea is to use the lower bound protocol described in the previous section to allow $\mathcal{P}$ to convince $\mathcal{V}$ that the cut is at least a certain size.

We first describe the protocol for proving a lower bound on the cut size in an unweighted graph. Given a cut $[S,T]$, for each vertex v, let $A_v = 1$ if $v \in S$ and $A_v = 0$ if $v \in T$.

```
𝒫 writes down A_v for each vertex v
𝒱 and 𝒫 perform the lower bound on set
  size protocol for the set
  C ≡ E ∩ {(u,v) | A_u ≠ A_v}
```

Note that $\mathcal{V}$ can determine membership in C in constant time. The weighted case may be treated by performing a lower bound on $\sum_{(u,v)\in C} w(u,v)$, where $w(u,v)$ is the weight of edge (u,v).

Obtaining a sublinear protocol in which $\mathcal{P}$ can convince $\mathcal{V}$ of a multiplicative approximation which upper bounds the size of a given cut is not possible: $\mathcal{V}$ requires $\Omega(n^2)$ time to distinguish between a cut size of 0 and 1, assuming the input graph is given in terms of its adjacency matrix.

MAXIMUM CONSTRAINT SATISFACTION PROBLEMS. Constraint satisfaction problems [Sch78, KST97] 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 $x_1, ..., x_n$ 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 which maximizes the sum of the weights of the satisfied constraints. The Max SAT problem and the Max Cut problem can both be cast as constraint satisfaction problems.

If $\mathcal{P}$ knows a solution of value v to the weighted max CSP problem, then $\mathcal{P}$ can convince a verifier $\mathcal{V}$ that the solution to the weighted Max CSP problem is at least $(1-\epsilon)v$ as follows: $\mathcal{P}$ initially writes down the $0/1$ settings of the x_i's. Then, using either of the protocols for showing approximate lower bounds on sums from the previous section, $\mathcal{P}$ convinces $\mathcal{V}$ that the sum of the weights of the satisfied constraints is at least $(1-\epsilon)v$. During the protocol, whenever $\mathcal{P}$ sends $\mathcal{V}$ pointers to constraints that are purportedly satisfied, $\mathcal{V}$ checks that the settings of the x_i's initially written by $\mathcal{P}$ satisfy the constraints.

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

MIN ONES CSPS. The *Min Ones CSP* problem involves finding a setting of the x_i's which minimizes the number of x_i's set to 1 and satisfies all of the constraints. It is easy to see that

Theorem 11 *There is a protocol by which $\mathcal{P}$ can convince $\mathcal{V}$ that there exists a setting of the x_i's which sets at most B of the x_i's to 1 and satisfies at least $1-\epsilon$ fraction of the constraints.*

We present the example of vertex cover of a graph with maximum degree d. This problem is NP-complete for any $d \geq 3$. Given graph $G = (V,E)$ of degree at most d with $|V| = n$, $|E| = m$, and a bound B, is there a set $C \subseteq V$ which is a vertex cover and $|C| \leq B$. If there is such a vertex cover, then there is a protocol by which $\mathcal{P}$ can convince $\mathcal{V}$ that there is a vertex cover of size at most $B(1+d\epsilon)$: $\mathcal{P}$ writes down an array of size at most B containing the vertex cover C. $\mathcal{V}$ chooses $O(1/\epsilon)$ edges and sends them to $\mathcal{P}$. $\mathcal{P}$ returns pointers to vertices in C which cover each of the edges. and $\mathcal{V}$ fails if some edge is not covered.

If C covers less than $(1-\epsilon)m$ edges, $\mathcal{V}$ is likely to fail. Otherwise, at most ϵm additional vertices will be required to cover the remaining ones. The claim follows since $m/d \leq B \leq m$.

A similar approach can be used for dominating set with a degree bound and set cover with bounded subset size, which are also NP-complete when the degree or set size bound is at least 3.

4.2 Constraint enforcement protocols

We have seen that designing protocols for proving lower bounds seems to be much easier than proving upper bounds. However, we show that a prover 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 [PST95]. We consider a *sparse* version of the problem where we are given $a_1, \ldots, a_n \geq 0$ and $b_{11}, \ldots, b_{nm} \geq 0$, 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 $x_i \geq 0$ and the m constraints $\forall j \in [m], \sum_i b_{ij} x_i \leq c_j$. 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 protocols. As long as there is an easy way to find all nonzero b_{ij} 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.

Now, $\mathcal{P}$ has a solution of value OPT in hand and wishes to convince $\mathcal{V}$ of the existence of a solution with value $\geq (1-\epsilon)$OPT which satisfies all of the constraints. To this end, we give a *constraint enforcement protocol*. All our results apply to the case when $\mathcal{P}$ has a solution of value v (not necessarily optimal) and would like to prove to $\mathcal{V}$ that the solution is of value at least $(1-\epsilon)v$.

4.2.1 Constraint enforcement: unweighted version

In order to describe the constraint enforcement protocol, 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 $b_{ij}x_i \in \{0,1\}$. $\mathcal{V}$ 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. $\mathcal{P}$ writes down the following *constraint enforcement structure* which consists of three parts: (i) For each constraint j, a list of the x_i's that are allocated "space" in constraint j (i.e., $b_{ij}x_i = 1$). More specifically, $\mathcal{P}$ writes *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 $b_{ij}x_i = 1$, then the capacity constraints are not violated. (ii) For each i, pointers to the locations in the constraint arrays in which x_i is allocated space, so that for each x_i set to 1, $\mathcal{V}$ can ensure that it is allocated space in each constraint j for which $b_{ij} > 0$. More specifically, a modification of permutation enforcement is used: $\mathcal{P}$ writes an array T of size at most n, such that $T[i] = \langle\langle j_1, \ell_1\rangle, \ldots, \langle j_t, \ell_t\rangle\rangle$ where $\langle j_a, \ell_a\rangle \in T[i]$ whenever x_i is present in constraint j_a and ℓ_a is that location in C_{j_a} such that $C_{j_a}[\ell_a] = i$. (iii) An array I of length n such that the i-th entry is the value of x_i.

Figure 1 shows the unweighted constraint enforcement protocol used for the following problem: Maximize $x_1 + x_2 + x_3 + x_4$ subject to $C_0 : x_1, x_2, x_3, x_4 \in \{0,1\}$, $C_1 : x_1 + x_2 \leq 1$, $C_2 : x_2 + x_3 + x_4 \leq 2$, and $C_3 : x_1 + x_3 \leq 1$. The solution setting $x_1 = x_4 = 1$ and $x_2 = x_3 = 0$ has value 2.

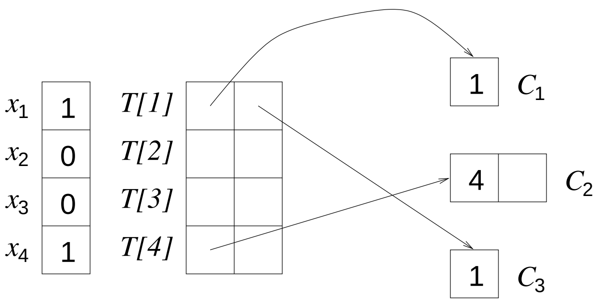

Figure 1: The unweighted case.

We say that element x_i is *good* if (i) $x_i = 1$ (i.e., $I(i) = 1$), (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 $G = \{i \mid x_i \text{ is good}\}$ be the set of good elements. Testing that i is good can be done in $O(t)$ time.

We do not want the verifier to check the entire structure or even an entire constraint; our gain comes from the fact that (i) the constraint enforcement structure ensures $\forall j \in [m], \sum_{i \in G} b_{ij}x_i \leq c_j$ and (ii) the value of the solution is at least the number of good elements. Setting $\hat{x}_i \leftarrow x_i$ for all $i \in G$ and $\hat{x}_i \leftarrow 0$ for all other i, we have:

Lemma 12 *$(\hat{x_1}, \ldots, \hat{x_2})$ is a feasible solution of value at least $|G|$.*

Thus the protocol is simply for $\mathcal{V}$ and $\mathcal{P}$ to run the protocol in Section 3.3 for showing that the size of the set $G = \{i \mid i \text{ is good}\}$ is at least $(1-\epsilon)\text{OPT}$, which in turn requires that $\mathcal{V}$ is able to check that a given i is good. The total runtime of $\mathcal{V}$ is $O(t/\epsilon)$.

Theorem 13 *There is a (t/ϵ)-approximate lower bound* PCPS *for unweighted fractional packing problems.*

4.2.2 Constraint enforcement: weighted version

We now consider the problem in the general form described above. We modify the previous protocol in two ways: (i) We modify the notion of *good* so that it is still the case that a solution $\hat{x}_1, \ldots, \hat{x}_n$ 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 $\sum_{x_i \in G} a_i x_i$. $\mathcal{V}$ can test whether i is good in $\text{polylog}(n)$ time. (ii) We use the protocols from Section 3.4 so that $\mathcal{V}$ can guarantee that $\sum_i a_i\hat{x}_i \geq (1-\epsilon)\text{OPT}$.

Since the values of the x_i's and their b_{ik} multipliers are no longer constrained to be either 0 or 1, we need to modify our method of keeping track of the "space" taken up by each nonzero $b_{ik}x_i$ in each constraint. 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. For each constraint, we maintain an array of length n, where the i-th entry records the running total of space taken up by the first i variables (we imagine the set to be a physical space of size c_i): the array C_j is $[r_1, r_2, r_3, \ldots, r_n]$ where $r_i = \sum_{k=1}^{i} b_{k,j} \cdot x_k$ 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 $\leq c_j$. Note that since the b_{ik}'s and x_i's are positive, if the r_j's are given correctly, they will form a monotone sequence.

Figure 2 shows the weighted constraint enforcement protocol used for the following problem: Maximize $x_1 + 2x_2 + 3x_3 + x_4$ subject to $C_0 : x_1, x_2, x_3, x_4 \geq 0$, $C_1 : x_1 + x_2 \leq 2$, $C_2 : x_2 + x_3 + x_4 \leq 4$, and $C_3 : x_1 + 2x_3 \leq 2$. The solution setting $x_1 = 0, x_2 = x_3 = 1, x_4 = 2$ has value 7.

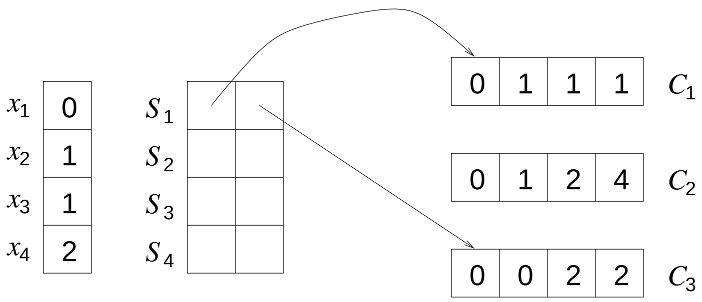

Figure 2: The weighted case.

In order to ensure that each constraint is satisfied, we need a definition of a *good* element which is strong enough so that (i) the sum of all good elements do not violate any constraints and (ii) $\mathcal{V}$ can efficiently determine whether an element is good (in particular, $\mathcal{V}$ does not have to look at many variables in the constraint). $\mathcal{P}$ could try to cheat $\mathcal{V}$ by giving a list of r_i's which is not monotone. However, if $r_{i_1-1} \leq r_{i_1} \leq r_{i_2-1} \leq r_{i_2} \ldots$ form a monotone subsequence, then it is easy to see that objects $i_1, i_2, \ldots$ can be simultaneously placed into the constraint without violating the capacity constraint. Our new definition of good borrows from the sorting spot-checker in [EKK+98]. An element r_i in a list of length m is said to be *heavy* if for all k, $0 \leq k \leq \lg m$, $r_i < r_j$ for at least 3/4 of the $j \in [i, i+2^k]$ and $r_i > r_j$ for at least $3/4$ of the $j \in [i-2^k, i]$ (for the purpose of defining heavy, if $i > m$ then r_i is assumed to be infinite, and if $i \leq 0$, r_i is assumed to be 0). In [EKK+98] it is shown that the heavy elements in a list form an increasing subsequence. Note that if both the corresponding r_i and r_{i-1} are heavy for each good element in a constraint, $r_0 \geq 0$ and r_i is less than the capacity of the constraint, then the sum of the good elements do not violate the constraint. Also note from the definition of a heavy element that it is possible to test the heaviness of an arbitrary element in $O(\lg n)$ time. We say an object i is *good* if for all $j \in S_i$: Let $r_i \leftarrow C_j[i]$, and $r_{i-1} \leftarrow C_j[i-1] = r_{i-1}$, then (i) $r_i - r_{i-1} = b_{i,j} x_i$, (ii) $0 < r_{i-1} < r_i \leq c_j$, and (iii) r_i and the preceding element r_{i-1} are both heavy with respect to the list $r_1, \ldots, r_n$. $\mathcal{V}$ can test if i is good in $O(t \lg n)$ time.

Lemma 14 *There is a $((t/\epsilon)\cdot\lg n)$-approximate lower bound* PCPS *for fractional packing problems.*

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 $\mathcal{P}$ knows how to construct a flow of size f, it is able to quickly prove to $\mathcal{V}$ that a flow of size $\geq (1-\epsilon)f$ exists. We sketch the method: To verify that a flow is legal, $\mathcal{V}$ must verify that the solution observes conservation of flow at each node and capacity constraints at each edge. $\mathcal{P}$ writes a list T of path-flows that combine to make up the flow of size f. $\mathcal{V}$ will pick random path-flows and ensure that they are "good" by checking that the flow is correctly packed into each edge that it follows – in doing so, $\mathcal{V}$ 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 upper bound enforcement structure ensures that no more than $c_{u,v}$ 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 unweighted version of the constraint enforcement protocol can be used to give a protocol by which the verifier can determine that there are enough good unit path flows in time $O(n/\epsilon)$. For larger flows, we use the result of [EK72] which shows that any flow can be decomposed into at most m (where m is the number of edges in the graph) path-flows. The weighted version of the constraint enforcement protocol can be used to give a protocol with runtime $O((n/\epsilon)\cdot\lg n)$.

The constraint enforcement structure can also be used to show a lower bound on the size of a multicommodity flow, where the runtime of $\mathcal{V}$ is $O((qn/\epsilon)\cdot\lg n)$, where q is the number of commodities.

BIN PACKING. n weighted objects, a bin size B, and an $\epsilon < 1$ are given. If it is possible to pack the objects into p bins, $\mathcal{P}$ can convince $\mathcal{V}$ that $p + \epsilon n$ bins are sufficient: $\mathcal{P}$ will use the constraint enforcement structure to assure $\mathcal{V}$ that at least $(1-\epsilon)$ fraction of the objects can be packed into p bins. The bound follows by placing the other objects into their own bins. $\mathcal{V}$'s running time is $O((1/\epsilon)\cdot\lg n)$.

EXACT COVER BY 3-SETS, MATCHING. Given set X with $|X| = 3q$ and a collection C of 3-element subsets of X. Does C contain an exact cover for X, i.e., a subcollection $E \subseteq C$ such that every element of X occurs in exactly one member of E? If so, then $\mathcal{P}$ can prove to $\mathcal{V}$ 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 $c = \{a_1, a_2, a_3\}$, c should appear in $C_{a_1}, C_{a_2}, C_{a_3}$. 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 $|F'| \geq (1-\epsilon)|F|$ and such that no $a \in X$ is covered more than once by F'. $\mathcal{V}$'s running time is $O(1/\epsilon)$.

Note that this construction works for any k (the runtime of $\mathcal{V}$ has linear dependence in k). In particular, since a matching is a cover by 2-sets, the protocol can be used to show an approximate lower bound PCPS 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 m products before deadline D, $\mathcal{P}$ can convince a $O((m/\epsilon)\lg p)$-time $\mathcal{V}$ that at least $(1-\epsilon)m$ products can be completed before the deadline: The protocol uses the 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, $\mathcal{P}$ can convince an $O((1/\epsilon)\lg n)$-time $\mathcal{V}$ that there exists a set S such that $B(1-\epsilon) \leq \sum_{i \in S} x_i \leq B$. A similar result holds for partition.

4.3 Matching problems

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

MATCHING. The following protocol is used to show a lower bound on the size of a matching: $\mathcal{P}$ writes a list L of edges in the matching. $\mathcal{P}$ convinces $\mathcal{V}$ that $|L \cap E| \geq (1 - \epsilon/2)k$. Then, $\mathcal{V}$ verifies that $\leq \epsilon/2$ fraction of edges involve vertices that are matched twice. To do this, $\mathcal{V}$ chooses a random edge $L[i] = (u, v)$ from L, then chooses a random vertex from $\{u, v\}$ and sends it to $\mathcal{P}$. $\mathcal{P}$ responds with the location j in L. $\mathcal{V}$ accepts if $i = j$. Thus there is a $(1/\epsilon)$-approximate lower bound PCPS protocol 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 $d \geq 3$. If there is such a matching, then there is a protocol by which $\mathcal{P}$ can convince an $O(1/\epsilon)$ time $\mathcal{V}$ that there is a maximal matching of size at most $U + n\epsilon/2$: $\mathcal{P}$ writes down the edges of a matching into an array of size U. To check that this matching is maximal, $\mathcal{V}$ randomly samples nodes and makes sure that a sampled node, if unmatched, does not have any unmatched neighbors. If the number of unmatched nodes with unmatched neighbors is more than ϵn, $\mathcal{V}$ is likely to 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.

5 IPS for set problems

We consider simple set problems of set intersection and set union and show protocols by which $\mathcal{P}$ can convince $\mathcal{V}$ of bounds on the result of these set operations. One application of these protocols is to proving bounds on the sizes of unions and intersections of databases queries.

TWO SET INTERSECTION. Consider the simpler version of the set intersection problem: Given sets A and B of cardinality n, and parameters ρ, is $|A \cap B|$ approximately ρn? Our interactive protocol will be given as input the sets A and B of cardinality n, and parameters ϵ, ρ, and will determine whether $(\rho - \epsilon)n \leq |A \cap B| \leq (\rho + \epsilon)n$.

Without the aid of the prover, the task requires $\Omega(\sqrt{n})$ time (cf. [EKK+98]). The lower bound protocol of [GS86] can be adapted to this setting to get multiplicative approximations of a lower bound, but we know of no such way to get a multiplicative approximation for the upper bound using the methods of [For89]. Our techniques can be viewed as special cases of their techniques, where the identity function is used in place of a hash function. The protocol for $\mathcal{V}$ is as follows:

```
P sends ε to V
Repeat the following m times:
  lower bound protocol
  V picks C ∈_R {A,B}
  V picks x ∈_R C and sends x to P
  P returns C' ∈ {A,B} and a pointer to
    x in C'
  If the pointer is valid and C = C',
    V sets k = k+1

  upper bound protocol
  V picks x ∈_R A and sends x to P
  if possible P returns a pointer to
    x in B
  If the pointer is valid,
    V sets ℓ = ℓ+1

V sets γ = k/m and δ = ℓ/m
Pass if 1−γ > ρ/2−ε and δ < ρ+ε
```

For the lower bound, first observe that if the intersection is large, $\mathcal{P}$ cannot but err on a lot of input elements. Given $x \in A \cap B$, the probability that $\mathcal{P}$ agrees with $\mathcal{V}$ is $1/2$. If $x \notin A \cap B$, since $\mathcal{P}$ is required to return a valid pointer, at best it agrees with $\mathcal{V}$ on all occasions. So, the probability that $\mathcal{P}$ agrees with $\mathcal{V}$ can be at most

$$\frac{1}{2}\frac{|A \cap B|}{n} + \frac{n - |A \cap B|}{n} = 1 - \frac{|A \cap B|}{2n}$$

$\mathcal{V}$, therefore, estimates a lower bound on this probability and hence an upper bound on $|A \cap B|/(2n)$ via γ. By Chernoff bounds, we can show that this estimate can be done to within an additive factor of ϵ by $\mathcal{V}$ with probability of error at most $O(\exp(-m^2))$.

For the upper bound, $\mathcal{P}$ can return a valid pointer only with probability of at most $|A \cap B|/n$, which can be upper bounded to within a factor of ρ by $\mathcal{V}$ via sampling. By Chernoff bounds, the probability of error is at most $O(\exp(-m^2))$.

Theorem 15 *For any $\rho > 0$, two-set-intersection has an $(1/(\epsilon\rho))$-interactive proof with $\Delta(y, y') = |y - y'|$.*

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

GENERAL SET INTERSECTION AND UNION. This also gives interactive protocols for checking, given $A_1, \ldots, A_k$ if $\cap_{i=1}^k A_i$ is large: $\mathcal{V}$ picks $i \in_R [k]$ and then $x \in_R A_i$ and sends x to the prover. $\mathcal{P}$ returns k pointers to location of x

(if valid) in each of A_i's. The details of these protocols are omitted in this version.

6 Interactive spot-checking

We have already mentioned two examples — element distinctness and set intersection — in which ISCs are provably faster than spot-checkers. A third example is from property testing[1] of the bipartiteness of graphs. This can be property tested in $\mathrm{poly}(1/\epsilon)$ time if the graph is represented as an adjacency matrix, and the distance ϵ is measured in terms of the fraction of entries that need to be changed in order to make the graph bipartite [GGR98]. In the adjacency list representation, the distance is measured in terms of the fraction of edges that need to be deleted. It is known that $\Omega(\sqrt{n})$ time is required for property testing [GR97] (an almost matching upper bound is given in [GR98]). It is easy to see that there is an ISC with runtime $\mathrm{poly}(1/\epsilon)$ for both representations.

Finally, the best known spot-checker for associativity runs in $\tilde{O}(n^{1.5})$ time [EKK+98]. Using the results of Section 5, one can give an ISC for associativity whose runtime is $\tilde{O}(n)$.

References

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems, *J. of the ACM*, 45(3):501–555, 1998.

[AS98] S. Arora and S. Safra. Probabilistic checkable proofs: A new characterization of NP. *J. of the ACM*, 45(1):70–122, 1998.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols, *Computational Complexity*, pp. 3–40, 1991.

[BFLS90] L. Babai, L. Fortnow, C. Lund, and M. Szegedy. Checking computations in polylogarithmic time. *Proc. 31st Foundations of Computer Science*, pp. 16–25, 1990.

[BGKW88] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. *Proc. 20th Symposium on Theory of Computing*, pp. 113–131, 1988.

[BK95] M. Blum and S. Kannan. Designing programs that check their work. *J. of the ACM*, 42(1):269–291, 1995.

[BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. of Computing and System Sciences*, 47(3):549–595, 1993.

[CMS99] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. Manuscript, 1999.

[CLSY90] J. Y. Cai, R. Lipton, R. Sedgewick, and A. Yao. Towards uncheatable benchmarks. *Proc. 8th Structure in Complexity Theory*, pp. 2–11, 1993

[CEG95] R. Canetti, G. Even, and O. Goldreich. Lower bounds for sampling algorithms for estimating the average. *IPL*, 53(1):17–25, 1995.

[CGKS95] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *Proc. 36th Foundations of Computer Science*, pp. 41–50, 1995. To appear in *J. of the ACM*.

[Con91] A. Condon. Space bounded probabilistic game automata. *J. of the ACM* 38(2):472–494, 1991.

[DKLR95] P. Dagum, R. Karp, M. Luby, and S. Ross. An optimal algorithm for Monte-Carlo estimation. *Proc. 36th Foundations of Computer Science* pp. 142–149, 1995.

[DS92] C. Dwork and L. Stockmeyer. Finite state verifiers I: The power of interaction. *J. of the ACM*, 39(4):800–828, 1992.

[1] Property testing problems as defined in [RS96, GGR98] can be cast as spot-checking problems.

[EK72] J. Edmonds and R. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. of the ACM*, 19(2):248–264, 1972.

[EKK+98] F. Ergun, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *Proc. 30th Symposium on Theory of Computing*, pp. 259–268, 1998.

[FGL+96] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques, *J. of the ACM*, 43(2):268–292, 1996.

[For89] L. Fortnow. The complexity of perfect zero-knowledge. *Randomness and Computation*, 5:327–343, 1989.

[FL93] L. Fortnow and C. Lund. Interactive proof systems and alternating time-space complexity. *Theoretical Computer Science*, 113:55–73, 1993.

[FRS94] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545–557, 1994.

[FS88] L. Fortnow and M. Sipser. Interactive proof systems with a log space verifier. Manuscript, 1988.

[GLR+91] P. Gemmell, R. Lipton. R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. *Proc. 23rd Symposium on Theory of Computing*, pp. 32–42, 1991.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *J. of the ACM*, 45(4):653–750, 1998.

[GR97] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Proc. 28th Symposium on Theory of Computing*, pp. 406–415, 1997.

[GR98] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Proc. 29th Symposium on Theory of Computing*, pp. 289–298, 1998.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. on Computing*, 18(1):186–208, 1989.

[GS86] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. *Proc. 18th Symposium on Theory of Computing*, pp. 59–68, 1986.

[KST97] S. Khanna, M. Sudan, and L. Trevisan. Constraint satisfaction: The approximability of minimization problems. *Proc. Symposium on Structure in Complexity Theory*, pp. 282–296, 1997.

[Kil] J. Kilian. Zero-knowledge with logspace verifiers. *Proc. 29th Foundations of Computer Science*, pp. 25–35, 1988.

[Kil92] J. Kilian. A note on efficient zero-knowledge proofs and arguments. *Proc. 24th Symposium on Theory of Computing*, pp. 723–732, 1992.

[Kil94] J. Kilian. Improved efficient arguments (preliminary version). *Proc. Advances in Cryptology – CRYPTO*, Springer LNCS 963:311–324, 1995.

[KO97] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. *Proc. 38th Foundations of Computer Science*, pp. 364–373, 1997.

[Lip91] R. Lipton. New directions in testing. *Proc. DIMACS Workshop on Distr. Comp. and Cryptography*, pp. 191–202, 1991.

[LFKN90] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems, *J. of the ACM*, 39(4):859–868, 1992.

[Mer90] R. C.Merkle. A certified digital signature. *Proc. Advances in Cryptology – CRYPTO*, Springer LNCS 435:218–238, 1989.

[Mic94] S. Micali. CS proofs. *Proc. 35th Foundations of Computer Science*, pp. 436–453, 1994.

[PST95] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Proc. 32nd Foundations of Computer Science*, pp. 495–504, 1991.

[RS96] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials and their applications to program testing. *SIAM J. on Computing*, 25(2):252–271, 1996.

[Sch78] T. Schaefer. The complexity of satisfiability problems. *Proc. 10th Symposium on Theory of Computing*, 1978.

[Sha90] A. Shamir. IP=PSPACE. *J. of the ACM*, 39(4):869–877, 1992.

[Spi96] D. A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Trans. on Information Theory*, 42(6):1723–1732, 1996.