Fast Approximate PCPs for Multidimensional Bin-Packing Problems ⋆

Tuğkan Batu [1]

School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6

Ronitt Rubinfeld [2]

CSAIL, MIT, Cambridge, MA 02139, USA

Patrick White

Computer Science Department, Cornell University, Ithaca, NY 14853, USA

Abstract

We consider approximate PCPs for multidimensional bin-packing problems. In particular, we show how a verifier can be quickly convinced that a set of multidimensional blocks can be packed into a small number of bins. The running time of the verifier is bounded by $O(\log^d n)$ where n is the number of blocks and d is the dimension.

Key words: Proof-assisted property testing; Probabilistically checkable proofs; Multidimensional bin-packing problems; Sublinear-time algorithms

⋆ Preliminary version appeared in the 3rd International Workshop on Randomization and Approximations Techniques in Computer Science, Random '99. This work was partially supported by ONR N00014-97-1-0505, MURI, NSF Career grant CCR-9624552, and an Alfred P. Sloan Research Award. The third author was supported in part by an ASSERT grant.

Email addresses: batu@cs.sfu.ca (Tuğkan Batu), ronitt@csail.mit.edu (Ronitt Rubinfeld), white@cs.cornell.edu (Patrick White).

[1] This research was done while the first author was a graduate student in the Department of Computer Science at Cornell University.

[2] Part of this research was done while the second author was visiting at IBM Almaden Research Center.

1 Introduction

Consider a scenario in which a user needs the optimal solution to a very large combinatorial optimization problem. The user asks another computational entity that has more resources to find a solution. The user then would like to trust that the *value* of the solution is feasible by only inspecting a very small portion of the solution. For example, suppose the services of a trucking company are needed by a mail-order company to handle all shipping orders. The mail-order company wants to ensure that the trucking company has the capacity to handle the orders. In this case, large amounts of typical data on the shipping loads might be presented to a computational entity to determine whether or not the load can be handled.

The probabilistically checkable proof (PCP) techniques (c.f., [1–3]) yield ways of formatting proofs so that their correctness can be verified quickly, even by inspecting only a constant number of bits of the proof. We note that the verifiers in the PCP results all require linear time in the size of the statement being proved. Approximate PCPs were introduced in [4] to allow a verifier to ensure that a solution to the optimization problem is at least *almost* correct when the input data is very large, and even linear time is prohibitive for the verifier. Approximate PCPs running in logarithmic or even constant time have been presented in [4] for several combinatorial problems. For example, a proof can be written in such a way as to convince a constant-time verifier that there exists a bin packing which packs a given set of objects into a small number of bins. Other examples include proofs which show the existence of a large flow, a large matching, or a large cut in a graph to a verifier that runs in sublinear time.

Our Results. We consider approximate PCPs for multidimensional bin-packing problems. In particular, we show how a verifier can be quickly convinced that a set of multidimensional objects can be packed into a small number of bins. Our results generalize the one-dimensional bin-packing results of [4].

The approximate PCP protocols for the bin-packing problem are more intricate in higher dimensions; for example, the placements and orientations of the blocks within the bin must be considered more carefully. In order to ensure that the placements of the blocks are nonoverlapping, we make use of properties that are related to monotonicity properties of functions defined on partially ordered sets. For now, in order to describe our results, we will not elaborate on the precise relationship, but we later give a careful description in the body of the paper. In the one-dimensional case, the approximate PCP protocol of [4] makes use of a property called *heaviness* of an element in a list, introduced

by [5]: View a list as a function f, such that the i^{th} element in a list is $f(i)$. Then, essentially, the *heaviness* of index i with respect to function f is defined so that testing if i is heavy can be done very efficiently (in logarithmic time in the size of the list) and such that the function values of all heavy indices in the list are necessarily in monotone increasing order. There is a natural generalization of monotonicity to functions over higher dimensions: for $x, y \in [n]^d$ such that $x \prec y$ (i.e., $x_i \leq y_i$ for all $i \in \{1, \ldots, d\}$), x and y are in *monotone order* if $f(x) \leq f(y)$. To ensure that the blocks are nonoverlapping in the multidimensional case, we generalize the notion of heaviness, giving heaviness properties and corresponding tests which determine the heaviness of a point $x \in [1, \ldots, n]^d$ in time $O(2^d \log^d n)$. Then, given a heaviness tester which runs in time $T(n)$, we show how to construct an approximate PCP protocol for bin packing in which the running time of the verifier is $O(T(n))$.

By definition, the heaviness of all domain elements with respect to a function f is equivalent to the monotonicity of function f. One can show that testing heaviness of randomly selected domain elements gives a test for the overall monotonicity of a function. Given function f from $[1, \ldots, n]^d$ to $[1, \ldots, r]$, a multidimensional monotonicity tester passes functions f that are monotone and fails functions f if no way of changing the value of f at less than ϵ fraction of the inputs will turn f into a monotone function. In [6], a monotonicity tester with query complexity $\tilde{O}(d^2 n^2 r)$ is given. Our multidimensional heaviness tester can also be used to construct a multidimensional monotonicity tester with the same asymptotic running time as the heaviness tester, that is, $O(2^d \log^d n)$. However, Dodis et al. [7] independently give monotonicity testers that are as efficient as ours for two dimensions and greatly improve on our running times for dimensions greater than two. The query complexity of their algorithm is $O(d \log(n) \log(r))$. More recently, the problem of testing the monotonicity of functions defined over general poset domains is studied in [8].

Halevy and Kushilevitz [9] have proposed a distribution-free property testing model, where the distance between functions is defined with respect to an arbitrary distribution over the domain from which the tester can take samples (as opposed to the uniform distribution). They have shown that a constant number of repetitions of one of our heaviness testers yields a monotonicity tester in the distribution-free property testing model. To our knowledge, there is no such analysis relying on any of the other known monotonicity testers.

2 Preliminaries

Notation. We use the notation $x \in_R S$ to indicate x is chosen uniformly at random from the set S. The notation $[n]$ indicates the interval $[1, \ldots, n]$.

We define a partial ordering relation $\prec$ over integer lattices such that if x and y are d-tuples then $x \prec y$ if and only if $x_i \leq y_i$ for all $i \in \{1, \ldots, d\}$. We use the notation $[x, y]$ to denote the set of all points z such that $x \prec z \prec y$. Consider a function $f : [n]^d \rightarrow \mathcal{R}$, where range $\mathcal{R}$ is a totally ordered set (order relation denoted by $\leq$). For $x, y \in [n]^d$ such that $x \prec y$, we say that x and y are in *monotone order* if $f(x) \leq f(y)$. We say f is *monotone* if for all $x, y \in [n]^d$ such that $x \prec y$, x and y are in monotone order.

Approximate PCP. The approximate PCP model is introduced in [4]. In this model, a verifier has query access to a theorem and a possibly-valid proof Π. It can make queries to Π in order to determine whether the theorem is close to a true theorem. More specifically, if on input x, a proof claiming that $f(x) = y$ is provided, the verifier wants to know if y is at least close to $f(x)$. As we will construct an approximate PCP protocol for a maximization problem, the following variant of the model will be used.

Definition 1 ([4]) *A function f is said to have a $t(\epsilon, n)$-time, $s(\epsilon, n)$-space, ϵ-approximate lower bound Probabilistically Checkable Proof system if there is a randomized verifier V with oracle access to the words of a proof Π 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 $y = f(x)$, then there is a proof Π of size $O(s(\epsilon, n))$ such that V^{Π} outputs PASS with probability at least $3/4$ (over the internal coin tosses of V);
(2) If $(1-\epsilon)y > f(x)$, then for all proofs Π', $V^{\Pi'}$ outputs FAIL with probability at least $3/4$ (over the internal coin tosses of V); and
(3) V runs in $O(t(\epsilon, n))$ time.

The verifier is a RAM machine which can read a word in one step. The probabilistically checkable proof protocol can be repeated $O(\log 1/\delta)$ times to get confidence at least $1 - \delta$. The analysis of our bin-packing protocol will show that if a proof claims to be able to pack all of the n input objects, the verifier can trust that at least $(1 - \epsilon)n$ of the objects can be packed.

It also follows from considerations in [4] that the protocols we give can be employed to prove the existence of suboptimal solutions. In particular, for a solution of value v, a proof for the existence of a solution of value at least $(1 - \epsilon)v$ can be written down. Since v is not necessarily the value of the optimal solution, these protocols can be used to trust the computation of approximation algorithms to the NP-complete problems we treat. This is a useful observation since the provider of the proof may not have computational powers outside of deterministic polynomial time, but might employ very good heuristics to get surprisingly good, yet not necessarily optimal, solutions.

Heaviness Testing. Our methods rely on the ability to define an appropriate *heaviness* property H on the domain elements of a function f. The heaviness property must be defined so that testing if a domain element is heavy with respect to H and f can be done very efficiently in the size of the domain, and such that all pairs of heavy elements in the domain that are comparable according to $\prec$ are in monotone order. Once the above is satisfied, our PCP protocols do not rely on the particular details of the heaviness property. Thus, we separate the discussion of the particulars of the heaviness properties and their associated tests (Section 4) from the way they are used in the PCPs (Section 3). As was mentioned earlier, heaviness properties are interesting in their own right, as their testers can be easily turned into distribution free monotonicity testers [9]. In the following, we describe the requirements of a heaviness property in more detail.

Before giving the precise requirements, let us begin by giving an example of a one-dimensional heaviness property and its corresponding test from [5]. We note that there were two such properties given in that work, both used to test if a list $L = (x_1, x_2, \ldots, x_n)$ is mostly sorted. Here we assume that the list contains distinct elements (a similar test covers the nondistinct case). The heaviness property is as follows: a list element x_i is *heavy* if a binary search on L for the value of x_i finds x_i at location i, without encountering any ordering inconsistencies along the search path. It is not hard to see that if two list elements x_i and x_j are heavy according to this definition, then they are in correct sorted order (since they are each comparable to their least common ancestor in the search tree). Furthermore, a test for whether i is heavy can be implemented in $O(\log n)$ time.

The definition of a heaviness property is generalized in this paper. We call a property a *heaviness property* if it implies that points with that property are in monotone order.

Definition 2 *Given a domain $\mathcal{D} = [1, \ldots, n]^d$, a function $f : \mathcal{D} \to \mathcal{R}$ and a property H over $\mathcal{D}$, we say that H is a* heaviness property *with respect to f if*

(1) $\forall x \prec y$, $H(x) \wedge H(y)$ implies $f(x) \leq f(y)$; and
(2) In a monotone function all points have property H.

If a point has a heaviness property H then we say that point is *heavy* (and omit referring to H when it is clear from the context). There may be many properties of points of a domain that are valid heaviness properties with respect to a given function. A challenge of designing good heaviness properties is to find properties which can be tested efficiently. A heaviness test is a probabilistic procedure which decides the heaviness property with high probability. If a point is not heavy, it should fail this test with high probability, and if a function is perfectly monotone, then every point should pass. Yet it is possible

that a function is not monotone, but a tested point is actually heavy. In this case the test may either pass or fail.

Definition 3 *Let $f : \mathcal{D} \to \mathcal{R}$ be a function on the domain $\mathcal{D} = [1, \ldots, n]^d$, and H be a heaviness property with respect to f. Let $S(\cdot, \cdot)$ be a randomized decision procedure on $\mathcal{D}$ with oracle access to function f. Given security parameter δ, we say S is a* heaviness test *for a heaviness property H if for all x, we have*

(1) If for all $z \prec y$, $f(z) \leq f(y)$, then $S^f(x, \delta) = \text{PASS}$; and
(2) If $H(x) \equiv \text{FALSE}$ (i.e., x is not heavy with respect to f), then

$$\Pr[S^f(x, \delta) = \text{FAIL}] \geq 1 - \delta.$$

We will use heaviness tests to build an approximate PCP verifier for the bin-packing problem. In particular, the verifier will use heaviness tests to enforce, among other properties, local multidimensional monotonicity of certain functions provided by the proof. It turns out that multidimensional heaviness testing is more involved than the one-dimensional version considered in earlier works, and raises a number of interesting questions. Our results on testing bin-packing solutions are valid for any heaviness property, and require only a constant number of applications of a heaviness test. We give sample heaviness properties and their corresponding tests in Section 4, yet it is an open question whether heaviness properties with more efficient tests exist. Such tests would immediately improve the efficiency of our approximate PCP verifier for bin packing.

Permutation Enforcement. Our protocols will require us to verify whether a given list $y_1, \ldots, y_n$ is a permutation of $[n]$, namely, $y_i = f(i)$ for some permutation f. In [4], the following method is suggested: The prover writes an array A of length n. $A(j)$ should contain i when $f(i) = j$ (its preimage according to f). We say that i is *honest* if $A(f(i)) = i$ (and, in particular, $0 \leq f(i) \leq n$). Note that the number of honest elements in $[n]$ lower bounds the number of distinct elements in $y_1, \ldots, y_n$ (even if A is written incorrectly). Thus, sampling $O(1/\epsilon)$ elements and determining that all of them are honest suffices to convince the verifier that there are at least $(1 - \epsilon)n$ distinct y_i's in $O(1/\epsilon)$ time. We refer to array A as the *permutation enforcer*.

3 Multidimensional Bin Packing

We consider the d-dimensional bin-packing problem. We assume that the objects to be packed are d-dimensional rectangular prisms, which we will here-

after refer to as blocks. The blocks are given as d-tuples (in $\mathbb{N}^d$) of their dimensions. Similarly, the bin size is given as a d-tuple, with entries corresponding to the integer width of the bin in each dimension. When we say a block with dimensions $w = (w_1, \ldots, w_d) \in \mathbb{N}^d$ is located at position $x = (x_1, \ldots, x_d)$, we mean that all the locations y such that $x \prec y \prec x + w - \vec{1}$, where $\vec{1}$ denotes all-ones vector, are occupied by this block. The problem of multidimensional bin packing is to try to find a packing of n blocks which uses the least number of bins of given dimension $D = (N_1, \ldots, N_d)$.

It turns out to be convenient to cast our problem as a maximization problem. We define the d-dimensional bin-packing problem as follows:

Input: n blocks, the dimensions of a bin, and an integer k
Output: a packing that packs the largest fraction of the blocks into k bins

It follows that if $1 - \epsilon$ fraction of the blocks can be packed in k bins, then at most $k + \epsilon n$ bins are sufficient to pack all of the blocks, by placing each of the remaining blocks in separate bins.

We give an approximate lower bound PCP protocol for the maximization version of the d-dimensional bin-packing problem in which the verifier runs in $O((1/\epsilon)T(N, d))$ time where $T(N, d)$ is the running time for a heaviness tester on $\mathcal{D} = [N_1] \times \cdots \times [N_d]$, and $N = \max_i N_i$. In all of these protocols, we assume that the block and bin dimensions fit in a word.

In this protocol, we assume that the verifier is provided with a proof that all the blocks can be packed in k bins. We require that the proof provides an encoding of a feasible packing of the input blocks in a previously-agreed format. This format is such that if all the input blocks can be packed in as few bins as claimed, the verifier accepts. If only less than $1 - \epsilon$ fraction of the input blocks can be simultaneously packed, the verifier rejects the proof with some constant probability. In the intermediate case, the verifier provides no guarantees.

3.1 A First Representation of a Packing

We represent a bin as a d-dimensional grid with the corresponding length in each dimension. The proof will label the packed blocks with unique integers and then label the grid elements with the label of the block occupying it in the packing. In Figure 1, we illustrate one such encoding.

The key to this encoding is that we give requirements by which the proof can define a monotone function on the grid using these labels only if there is a feasible packing. To show such a reduction exists, we first define a relation on

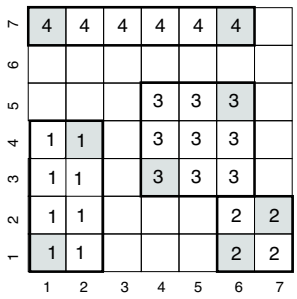

Fig. 1. A 2D Encoding

blocks.

Definition 4 *For a block b,* the highest corner of b, *denoted $h^{(b)}$, is the corner with the largest coordinates in the bin it is packed with respect to the $\prec$ relation. Similarly,* the lowest corner of b, *denoted $l^{(b)}$, is the corner with the smallest coordinates.*

In our figure, $l^{(1)} = (1, 1)$ and $h^{(1)} = (2, 4)$. We can order blocks by only considering the relative placement of these two corners.

Definition 5 *Let b_1 and b_2 be two blocks packed in the same bin. Block b_1* precedes *block b_2 in a packing if $l^{(b_1)} \prec h^{(b_2)}$.*

Note that for a pair of blocks in dimension higher than one it may be the case that neither of the two blocks precedes the other. This fact along with the following observation makes this definition interesting.

Observation 6 *For two blocks, b_1 and b_2, such that b_1 precedes b_2, b_1 and b_2 overlap if and only if b_2 precedes b_1.*

Surely if b_1 precedes b_2 and this pair overlaps it must be the case that $l^{(b_2)} \prec h^{(b_1)}$. It follows that the precedence relation on blocks is a reflexive antisymmetric ordering precisely when the packing of the blocks is feasible. Given such an ordering, it is easy to construct a monotone function.

Lemma 7 *Given a feasible packing of a bin with blocks, we can label the blocks with distinct integers such that when we assign each occupied grid element in the d-dimensional grid (of the bin) with the label of the block occupying it, we get a monotone partial function, which can be extended to monotone total function, on this grid.*

PROOF. The relation from Definition 5 gives a relation on the blocks that is reflexive and antisymmetric. Therefore we can label the blocks according to this relation such that a block gets a label larger than those of all its predecessors. This labeling gives us a monotone partial function on the grid. To

extend this partial function to a total function, each unoccupied grid element can be assigned the smallest possible value that precedes it. □

Now we can describe the proof. The proof will consist of three parts:

(1) A table that will have an entry for each block containing:
  (i) the label assigned to the block;
  (ii) a pointer to the bin to which the block is assigned; and
  (iii) the locations of the two (the lowest and the highest) corners of the block in this bin.
(2) A permutation enforcer on the blocks and the labels of the blocks.
(3) A d-dimensional grid with dimensions of size $N_1 \times \ldots \times N_d$ for each bin used that labels each grid element with the label of the block occupying it.

3.2 Testing Multidimensional Bin-Packing Solutions Using Heaviness

We present a verifier protocol for testing the previously described proof format. We assume a particular heaviness property H and its associated tester. Our tester will be based on showing that if all the defining corners of a pair of blocks are heavy, then they cannot overlap.

Protocol. We will define "good" blocks such that all good blocks can be packed together feasibly. Our notion of good will have the properties that (i) a good block is actually packed inside a bin such that it is not overlapping any other good block; and (ii) we can efficiently test a block for being good. Then, the verifier uses sampling to ensure that at least $1 - \epsilon$ fraction of the blocks are good.

Definition 8 *The block i with dimensions $w = (w_1, \ldots, w_d)$ is* good *with respect to an encoding of a packing and a heaviness property H if it has the following properties:*

- *The two corners defining the block in the proof have values inside the bin, i.e., $\vec{1} \prec l^{(i)} \prec h^{(i)} \prec \vec{N}$, where $\vec{N}$ stands for the highest corner of the bin.*
- *The distance between these corners exactly fits the dimensions of the block, i.e., $w = h^{(i)} - l^{(i)} + \vec{1}$.*
- *The grid elements at $l^{(i)}$ and $h^{(i)}$ are heavy according to property H with respect to the labeling of the grid elements.*
- *The block is assigned a unique label among the good blocks, that is, it is honest with respect to the permutation enforcer.*

Given this definition, we can prove that two good blocks cannot overlap.

Lemma 9 *If two blocks overlap in a packing, then both of the blocks cannot be good with respect to this packing.*

PROOF. Note that when two blocks overlap, according to Definition 5, they must both precede each other, that is, $l^{(b_1)} \prec h^{(b_2)}$ and $l^{(b_2)} \prec h^{(b_1)}$. We know, by the definition of a heaviness property, that two comparable heavy points on the grid do not violate monotonicity. Since both defining corners of a good block must have the same label, either $l^{(b_1)}$ and $h^{(b_2)}$, or $l^{(b_2)}$ and $h^{(b_1)}$ violates monotonicity. □

Corollary 10 *There is a feasible packing of all the good blocks in an encoding using k bins.*

The verifier's protocol can be given as follows:

Repeat $O(\frac{1}{\epsilon})$ times:
  Choose a block b uniformly at random
  Test if block b is good by
    checking $\vec{1} \prec l^{(b)} \prec h^{(b)} \prec \vec{N}$ and $w_b = h^{(b)} - l^{(b)} + \vec{1}$,
      where w_b is the dimension of b,
    checking unique labeling for b using permutation enforcer, and
    performing heaviness tests for $l^{(b)}$ and $h^{(b)}$.

The verifier, by testing $O(1/\epsilon)$ randomly chosen blocks, ensures that at least $(1-\epsilon)$ fraction of the blocks are good. Hence, we get the following theorem.

Theorem 11 *There is an $O((1/\epsilon)T(N,d))$-time, $O(nN^d)$-space, ϵ-approximate lower bound PCP for the d-dimensional bin packing problem where $T(N,d)$ is the running time for a heaviness tester on $\mathcal{D} = [N_1] \times \cdots \times [N_d]$, $N = \max_i N_i$, and n is the number of blocks in the input.*

3.3 A Compressed Representation of a Packing

The previous protocol requires a proof such that the size of the proof depends on the *dimensions* N_i of the bins to be filled. We show here how to write a proof such that the size of the proof depends only on the number of blocks to be packed. In the protocol from the previous section the verifier calls the heaviness tester only on grid elements that correspond to the lowest or the highest corners of the blocks. We use this observation to get a compressed proof.

The proof uses a set of *distinguished coordinate* values S_k for each dimension $k = 1, \ldots, d$. Here is how the S_k's are constructed. Each set S_k is initially empty. For each block i and for the lowest corner, $l^{(i)} = (c_1, \ldots, c_d)$, and the highest corner, $h^{(i)} = (e_1, \ldots, e_d)$, of block i, $S_k \leftarrow S_k \cup \{c_k\} \cup \{e_k\}$. After all the blocks are processed, $|S_k| \leq 2n$. The *compressed grid* will be a sublattice of $\mathcal{D}$ with each dimension restricted to these distinguished coordinates, that is the set $\{\langle x_1, \ldots, x_d \rangle | x_k \in S_k\}$. This grid will contain in particular all the corners of all the blocks and the size of the grid will be at most $O((2n)^d)$. The fact that this new compressed encoding is still easily testable does not trivially follow from the previous section. In particular, we must additionally verify that the compression is valid.

The proof consists of four parts. First the proof from the previous section, which we refer to as the *original grid*, is implicitly defined. The new proof consists of a table containing the *compressed grid*. In each axis, the coordinates are labeled by $[1, \ldots, 2n]$ and a *lookup-table* (of length $2n$) is provided for each axis which maps compressed grid coordinates to original grid coordinates. Finally a list of blocks with pointers to the compressed grid, and a permutation enforcer as before is provided. In Figure 2 , we give the compressed encoding of the packing from Figure 1.

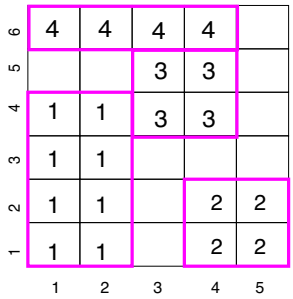

Fig. 2. A Compressed Encoding

Protocol. By enabling the compressed proof to contain only a portion of the proof from the first protocol, we provide more opportunities for a cheating proof. For example, even if the compressed proof uses the correct set of hyperplanes for the compression, it may reorder them in the compressed grid to hide overlapping blocks. The conversion tables we introduced to our proof will allow the verifier to detect such cheating.

The definition of a good block is extended to incorporate the lookup tables. In a valid proof, the lookup tables would each define a monotone function on $[2n]$. We will check that the entries in the lookup tables which are used in locating a particular block are *heavy* in their respective lookup tables. Additionally we test a that a block is good with respect to Definition 8 in the compressed

grid.[3] A block which passes both phases is a *good* block.

Our new protocol is then exactly as before. The verifier selects $O(1/\epsilon)$ blocks and tests that each is good, and if so, concludes that at least $1-\epsilon$ fraction of the blocks are good.

Correctness. Any two good blocks do not overlap in the compressed grid, by applying Lemma 9. Furthermore, since the labels of good blocks in the lookup table are heavy, it follows that two good blocks do not overlap in the original grid either. Certainly, since the corresponding values in the lookup table form a monotone sequence, the proof could not have re-ordered the columns during compression to untangle an overlap of blocks. It also follows from the earlier protocol that good blocks are the right size and are uniquely presented.

Theorem 12 *There is an $O((1/\epsilon)T(n,d))$-time, $O((2n)^{d+1})$-space, ϵ-approximate lower bound PCP for the d-dimensional bin-packing problem, where $T(n,d)$ is the running time for a heaviness tester on $\mathcal{D} = [2n]^d$ and n is the number of blocks in the input.*

3.4 An Extension to Recursive Bin Packing

At the simplest level the recursive bin-packing problem takes as input a set of blocks, a list of container sizes (of unlimited quantity), and a set of bins. Instead of placing the blocks directly in the bins, a block must first be fit into a container (along with other blocks) and the containers then packed in the bin. The goal is to minimize the total number of bins required for the packing. We can give a protocol by which a proof can convince a verifier that a good solution exists by applying an extension of our multidimensional bin-packing tester. In particular, we define a block as *good* if it passes the goodness test (with respect to its container) given in Section 3 and furthermore if the container it is in passes the same goodness test (with respect to the bin). After $O(1/\epsilon)$ tests we can conclude that most blocks are good and hence that $(1-\epsilon)$ fraction of the blocks can be feasibly packed. For a k-level instance of recursive bin packing, therefore, the proof will have k compressed proofs and $O(k/\epsilon)$ goodness tests will be needed.

[3] Except when we test the size of the block, for which we refer to the original coordinates via the lookup table.

3.5 Can Monotonicity Testing Help?

Given the apparent similarities between heaviness testing and monotonicity testing, it may seem that a monotonicity test could be used to easily implement our multidimensional bin-packing protocol. The obvious approach, though, does not seem to work. The complications arise because we are embedding n blocks in a $(2n)^d$ sized domain. If a monotonicity tester can determine that the domain of our compressed proof has $(1 - \epsilon')$ of its points in a monotone subset, we can only conclude that at least $n - \epsilon' \cdot (2n)^d$ boxes are "good", by distributing the bad points among the corners of the remaining boxes. Thus a direct application of monotonicity testing on this domain seems to need an error parameter of $O(\epsilon/(n^d))$. If the running time of the monotonicity tester is linear in ϵ then this approach requires at least $O((2n)^{d-1})$ time.

4 Tests for Two Heaviness Properties

In this section, we define two separate heaviness properties and provide their corresponding tests for functions over a domain isomorphic to an integer lattice. Both of these heaviness properties and their tests are generalizations of the results from [5] in a one-dimensional domain. We denote the domain of the functions as $\mathcal{D} = [1, \ldots, n]^d$. The range $\mathcal{R}$ of the functions can be any partial order. Both tests which follow can determine that a point is heavy in $O((2 \log n)^d)$ time. These running times yield efficient bin packing tests for small values of d as described in Section 3.

4.1 The First Heaviness Property

In order to define our first heaviness property, we consider a set of $\log^d n$ carefully chosen neighborhoods around a point x. At a high level, the point x has this property if for a large fraction of points y in each of these neighborhoods, x and y are in monotone order. We are able to show from this that for any two points $x \prec x'$ such that x, x' both have the property, two neighborhoods can be found, one around each point, whose intersection contains a point z with the property that $x \prec z \prec x'$ and $f(x) \leq f(z) \leq f(x')$. Hence this defines a valid heaviness property.

We consider the following graph induced by a function f over a partially ordered domain: The vertices in the graph correspond to points of the domain, while edges are inserted between points that are monotonically ordered according to f.

Definition 13 *The graph G_f induced by a function $f : \mathcal{D} \to \mathcal{R}$ is a directed graph where $V(G_f) = \mathcal{D}$ and $E(G_f) = \{(x, y) | x \prec y$ and $f(x) \leq f(y)\}$.*

Given this graph G_f, a point x, and a deviation h, we are interested in the number of points in the intervals $[x, x + h]$ and $[x - h, x]$ which are monotonically ordered with x according to f. In terms of G_f we want to know how many of the out-edges originating at x terminate in the subgraph of points $[x, x+h]$ and how many of the in-edges to x originate in the subgraph of points $[x - h, x]$. We define these points as functions of x and h.

Definition 14 *$\Gamma^+_{h_1,\ldots,h_n}(x)$ is the set of points y in the domain such that $x \prec y$, $y \prec x + h$, and $(x, y) \in E(G_f)$. Similarly, $\Gamma^-_{h_1,\ldots,h_n}(x)$ is the set of points y in the domain such that $y \prec x$, $x \prec y + h$, and $(y, x) \in E(G_f)$.*

Using these definitions, we can formalize the notion of a heavy point. Intuitively, these heavy points have lots of incoming and outgoing arcs from and to every neighborhood around them.

Definition 15 *A point x in the graph G_f (or, equivalently, in $\mathcal{D}$) is* η-good *if for all i, all integers k_i, $0 \leq k_i \leq \log x_i$, $|\Gamma^-_{2^{k_1},\ldots,2^{k_d}}(x)| \geq \eta 2^{\sum_i k_i}$, and for all integers k_i, $0 \leq k_i \leq \log(n - x_i)$, $|\Gamma^+_{2^{k_1},\ldots,2^{k_d}}(x)| \geq \eta 2^{\sum_i k_i}$.*

Note that our definition of η-good requires that x satisfy requirements over $O(\log^d n)$ subsets of $\mathcal{D}$. Now, we will instantiate η in the definition above to obtain a heaviness property H_1 over $\mathcal{D}$.

Definition 16 *A point $x \in \mathcal{D}$ has* property H_1 *if x is η-good for $\eta = 1 - 2^{-d-1}$.*

The following lemma states that H_1 is indeed a heaviness property.

Lemma 17 *Property H_1 is a heaviness property, that is, if $x \prec y$ and x and y have property H_1, then $(x, y) \in E(G_f)$.*

PROOF. Fix $x \prec y$. Consider the d-dimensional rectangular hyperprism of which x and y are the opposite endpoints, i.e., all points z such that $x_i \leq z_i \leq y_i$ for all i. Let I denote the space of points in $\mathcal{D}$ within this closed hyperprism. Let $\Delta_i = y_i - x_i$ denote the lengths of each of the axes of this hyperprism. Define m_i so that $\Delta_i \leq m_i < 2\Delta_i$ and $m_i = 2^{k_i}$ for some integer k_i. Now extend I to a new hyperprism S such that the lengths of the axes of S are given by the set of m_i's defined above, that is, intuitively "round up" each side length to the next power of two. By the definitions of H_1 and of S, we can now lower bound the number of edges from x into I in terms of $|S|$. Fix $\eta = 1 - 2^{-d-1}$. We can bound $|\Gamma^+_{m_1,\ldots,m_d}(x) \cap I|$ such that

$$|\Gamma^+_{m_1,\ldots,m_d}(x) \cap I| \geq \eta|S| - |S \setminus I| = \eta|S| - |S| + |I| = |I| - (1 - \eta)|S|,$$

and similarly,

$$|\Gamma^-_{m_1,\ldots,m_d}(y) \cap I| \geq |I| - (1-\eta)|S|.$$

If we can show that $|\Gamma^+_{m_1,\ldots,m_d}(x) \cap I| + |\Gamma^-_{m_1,\ldots,m_d}(y) \cap I| > |I|$ then the pigeonhole principle can be applied to find some vertex z with $(x,z) \in E(G_f)$ and $(z,y) \in E(G_f)$. By transitivity, we would have shown that $(x,y) \in E(G_f)$. We solve the equation given above to get

$$2\left(|I| - (1-\eta)|S|\right) > |I|$$

if and only if

$$|I| > 2(1-\eta)|S| = 2(2^{-(d+1)})|S| = 2^{-d}|S|.$$

This last line is true since $|I|$ and $|S|$ are d-dimensional and every side of S is less than twice the length of the corresponding side in I. $\square$

Now we can present the corresponding heaviness test. On input x, our test compares x to several random elements y selected from carefully chosen neighborhoods around x. It is tested that x is in order with a large fraction of points in each of these neighborhoods. The test is shown in Figure 3.

```
HeavyTest(f, x, δ)
  for  k_1 ← 0 ... log x_1,
       ⋮
       k_d ← 0 ... log x_d do
           repeat t = O(2^d log(1/δ)) times
               choose h_i ∈_R [1, 2^{k_i}]  1 ≤ i ≤ d
               h ← (h_1, ..., h_d)
               if (f(x) < f(x − h)) return FAIL
  for  k_d ← 0 ... log(n − x_1),
       ⋮
       k_d ← 0 ... log(n − x_d) do
           repeat t times
               choose h_i ∈_R [1, 2^{k_i}]  1 ≤ i ≤ d
               h ← (h_1, ..., h_d)
               if (f(x) > f(x + h)) return FAIL
  return PASS
```

Fig. 3. Algorithm HeavyTest

Theorem 18 *Algorithm HeavyTest is a heaviness tester for property H_1 with query complexity $O(\log(1/\delta)2^d \log^d n)$ and error probability δ.*

PROOF. Given a function f over $\mathcal{D}$ and a point x, this algorithm constructs $O(\log^d n)$ neighborhoods around x and explicitly checks (by sampling $O(2^d \log(1/\delta))$ times) that x have property H_1. For a monotone function f, it is clear that the algorithm always outputs PASS. For a point x that does not have property H_1, there must be a neighborhood of x that violates the heaviness condition. When the algorithm tests this neighborhood of x, it will sample a point y such that x and y are not in monotone order with probability at least $1 - \delta$. □

4.2 The Second Heaviness Property

In this section, we present a recursively-defined heaviness property. Namely, a point x is heavy in dimension d if a certain set of projections of x onto hyperplanes are each heavy in dimension $d-1$. We are able to use the heaviness of these projection points to conclude that d-dimensional heavy points are appropriately ordered.

Given a d-dimensional hypercube C, consider a subdividing operation ϕ which maps C into 2^d congruent subcubes. This operation passes d hyperplanes parallel to each of the axes of the hypercube through the center. We call each of these dividing hyperplanes a *bisector*. We also define $\Phi_x(C)$ as the unique $S \in \phi(C)$ that contains x. This function is also a notational convenience which identifies the subcube a point lies in after such a division. For nonnegative integer r, we recursively define $\Phi_x^r(C) = \Phi_x(\Phi_x^{r-1}(C))$ where $\Phi_x^1 = \Phi_x$.

Now consider any two distinct points in the hypercube, x and y. We wish to apply ϕ to the cube repeatedly until x and y are no longer in the same cube. To quantify this we define a new function $\varrho : C^2 \to \mathbb{Z}$ such that $\varrho(x, y) = r$ only when $\Phi_x^r(C) = \Phi_y^r(C)$ and $\Phi_x^{r+1}(C) \neq \Phi_y^{r+1}(C)$. That is, the $(r+1)$st composition of Φ on C separates x from y.

Definition 19 *A point x has* property H_2 *in a domain $\mathcal{D} = [n]^d$ if the d perpendicular projections of x onto each bisector of each cube in the series $\Phi_x(\mathcal{D}), \ldots, \Phi_x^{\log n}(\mathcal{D})$ of shrinking cubes all have property H_2 in dimension $d-1$. The domains for these recursive tests are the respective bisectors of the cubes. When $d = 1$, point x have property H_2 if it is $\frac{3}{4}$-good (according to Definition 15, Section 4.1).*

We can now present the corresponding heaviness test for a point. Let C be a d-dimensional integer hypercube with side length n. Let x be some point in C. Construct the sequence $\{S_1, \ldots, S_k\} = \{\Phi_x(C), \Phi_x^2(C), \ldots, \Phi_x^k(C)\}$ where $k = \lceil \log(n) \rceil$. Note that $S_k = x$. At each cube S_j, perform the following test: (i) Compute the d perpendicular projections $\{p_1, \ldots, p_d\}$ of x onto the d bisectors of S_{j-1}; (ii) Verify that f is consistent with a monotone function on each of

the d pairs (x, p_i); (iii) If $d > 1$, then recursively test that each of the points p_i have property H_2 over the reduced domain of its corresponding bisector in S_{j-1}. If $d = 1$, we use the heaviness test of Section 4.1. The algorithm is shown in Figure 4.

```
RecursiveHeavyTest(C, f, x, δ)
  let d = dimension of C
  if d = 1
      return HeavyTest(f, x, δ)
  else
      S_0 = C, S_i = Φ^i_x(C) for i = 1, ..., log n
      for i = 1, ..., log n
          {p_1, ..., p_d} = projections of x onto the bisectors of S_{i-1}
          for k = 1, ..., d
              Verify that x and p_k are in monotone order
              C' ← the bisector of S_{i-1} containing p_k
              RecursiveHeavyTest(C', f, p_k, δ)
  return PASS
```

Fig. 4. Algorithm RecursiveHeavyTest

Lemma 20 *Property H_2 is a heaviness property.*

PROOF (by induction on d). Let $r = \varrho(x, y)$. Let $S = \Phi^r_x(C)$. Let $S_x = \Phi^{r+1}_x(C)$ and $S_y = \Phi^{r+1}_y(C)$. There is at least one bisector of S which separates x and y. This plane also defines a face of S_x and of S_y. By induction, we know the projections of x and y onto these faces have property H_2. Since y dominates x in every coordinate, we know that $p_x \prec p_y$. Inductively, we can conclude from the heaviness of the projection points that $f(p_x) \leq f(p_y)$. Since we have previously tested that $f(x) \leq f(p_x)$ and $f(p_y) \leq f(y)$, we conclude $f(x) \leq f(y)$. □

Running Time Analysis Let $R_d(n)$ be the number of times that RecursiveHeavyTest calls HeavyTest algorithm when testing that a point of the function $f : \mathcal{D} \to \mathcal{R}$ have property H_2, then we can show

Lemma 21 *For all $d \geq 1$, for sufficiently large n, $R_d(n) \leq d \log^{d-1}(n)$.*

PROOF. We use proof by induction. For the case $d = 1$, we call HeavyTest algorithm directly. So, clearly, $R_1(n) = 1$. We now assume $R_d(n) \leq d \log^{d-1}(n)$, and prove $R_{d+1}(n)$ is as claimed. By expanding $R_{d+1}(n)$ recursively and using $\sum_{i=1}^{m} i^k \leq (m+1)^{k+1}/(k+1)$, we get

$$
\begin{aligned}
R_{d+1}(n) &= (d+1)(R_d(n) + R_d(n/2) + \cdots + R_d(2) + R_d(1)) \\
&\leq (d+1) \sum_{i=1}^{\log n} d \log^{d-1}(2^i) \\
&= d(d+1) \sum_{i=1}^{\log n} i^{d-1} \\
&\leq d(\log(n)+1)^d \\
&\leq (d+1)\log^d(n)
\end{aligned}
$$

for $\log n \geq d^2$. $\square$

Theorem 22 *Algorithm RecursiveHeavyTest is a heaviness tester for the heaviness property in Definition 19 with query complexity* $O(d \log^d(n) \log(1/\delta))$ *and error probability* δ.

PROOF. It is clear from the properties of HeavyTest that RecursiveHeavyTest always outputs PASS when the input function f is monotone. For a point x that does not have property H_2, there must exist a projection x' of x that does not have property H_2 in one dimension. Since HeavyTest will be called on x', RecursiveHeavyTest outputs FAIL with probability at least $1-\delta$. Since the query complexity of HeavyTest in one dimension is $O(\log(1/\delta)\log(n))$, the query complexity of RecursiveHeavyTest is $O(d \log^d(n) \log(1/\delta))$ by Lemma 21. $\square$

Acknowledgements

We thank the anonymous referees for their helpful suggestions on improving the presentation of the paper.

References

[1] L. Babai, L. Fortnow, C. Lund, Non-deterministic exponential time has two-prover interactive protocols, Computational Complexity 1 (1991) 3–40.

[2] L. Babai, L. Fortnow, L. A. Levin, M. Szegedy, Checking computations in polylogarithmic time, in: Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, New Orleans, Louisiana, 1991, pp. 21–31.

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

[4] F. Ergun, R. Kumar, R. Rubinfeld, Fast approximate probabilistically checkable proofs, Information and Computation 189 (2) (2004) 135–159.

[5] F. Ergun, S. Kannan, R. Kumar, R. Rubinfeld, M. Viswanathan, Spot-checkers, Journal of Computer and System Sciences 60 (3) (2000) 717–752.

[6] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, Testing monotonicity, in: Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, Los Alamitos, CA, 1998, pp. 426–435.

[7] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, A. Samorodnitsky, Improved testing algorithms for monotonicity, in: D. Hochbaum, K. Jensen, J. D. Rolim, A. Sinclair (Eds.), Randomization, Approximation, and Combinatorial Optimization, Vol. 1671, LNCS, Berkeley, California, 1999, pp. 96–108.

[8] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, A. Samorodnitsky, Monotonicity testing over general poset domains, in: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, Montréal, Québec, Canada, 2002, pp. 474–483.

[9] S. Halevy, E. Kushilevitz, Distribution-free property testing, in: RANDOM: International Workshop on Randomization and Approximation Techniques in Computer Science, Vol. 1671, LNCS, Princeton, NJ, 2003, pp. 302–317.