Fast Approximate PCPs for Multidimensional Bin-Packing Problems ⋆

Tuğkan Batu[1], Ronitt Rubinfeld[1]⋆⋆, and Patrick White[1]

Department of Computer Science, Cornell University, Ithaca, NY 14850
{batu,ronitt,white}@cs.cornell.edu

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 $O(T(n))$, where $T(n)$ is the time required to test for *heaviness*. We give heaviness testers that can test heaviness of an element in the domain $[1, \ldots, n]^d$ in time $O((\log n)^d)$. We also also give approximate PCPs with efficient verifiers for recursive bin packing and multidimensional routing.

1 Introduction

Consider a scenario in which the optimal solution to a very large combinatorial optimization problem is desired by a powerful corporation. The corporation hires an independent contractor to actually find the solution. The corporation then would like to trust that the *value* of the solution is feasible, but might not care about the structure of the solution itself. In particular they would like to have a quick and simple test that checks if the contractor has a good solution by only inspecting a very small portion of the solution itself. Two hypothetical situations in which this might occur are:

– A major corporation wants to fund an international communications network. Data exists for a long history of broadcasts made over currently used networks, including bandwidth, duration, and integrity of all links attempted. The corporation wants to ensure that the new network is powerful enough to handle one hundred times the existing load.
– The services of a trucking company are needed by an (e-)mail-order company to handle all shipping orders, which involves moving large numbers of of boxes between several locations. The mail-order company wants to ensure that the trucking company has sufficient resources to handle the orders.

In both cases, large amounts of typical data are presented to the consulting company, which determines whether or not the load can be handled. The probabilistically checkable-proof (PCP) techniques (cf. [3, 4, 1]) offer ways of verifying such solutions quickly. In

⋆ 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.
⋆⋆ Part of this work was done while on sabbatical at IBM Almaden Research Center

these protocols a proof is written down which a verifier can trust by inspecting only a constant number of bits of the proof. The PCP model offers efficient mechanisms for verifying any computation performed in NEXP with an efficient verifier. We note that the verifiers in the PCP results all require $\Omega(n)$ time. Approximate PCPs were introduced in [7] for the case when the input data is very large, and even linear time is prohibitive for the verifier. Fast approximate PCPs allow a verifier to ensure that the answer to the optimization problem is at least *almost* correct. Approximate PCPs running in logarithmic or even constant time have been presented in [7] 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. 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. We also consider the related problems of recursive bin packing and multidimensional routing. Our results generalize the 1-dimensional bin packing results of [7]. The PCPs are more intricate in higher dimensions; for example, the placements and orientations of the blocks within the bin must be considered more carefully. In the 1-dimensional case, the approximate PCP of [7] makes use of a property called *heaviness* of an element in a list, introduced by [6]. Essentially, *heaviness* is defined so that testing if an element is heavy can be done very efficiently (logarithmic) in the size of the list and such that all heavy elements in the list are in monotone increasing order. We generalize this notion to the multidimensional case and give heaviness tests which determine the heaviness of a point $x \in [1, \ldots, n]^d$ in time $O((2 \log n)^d)$. Then, given a heaviness tester which runs in time $T(n)$, we show how to construct an approximate PCP for binpacking in which the running time of the verifier is $O(T(n))$.

In [9], multidimensional monotonicity testers are given which pass functions f that are monotone and fail functions f if no way of changing the value of f at at most ϵ fraction of the inputs will turn f into a monotone function. The query complexity of their tester is $\tilde{O}(d^2 n^2 r)$ where f is a function from $[n]^d$ to $[r]$. Our multidimensional heaviness tester can also be used to construct a multidimensional monotonicity tester which runs in time $O(T(n))$. However, more recently Dodis *et. al.* [5] have given monotonicity testers that greatly improve on our running times for dimension greater than 2, and are as efficient as ours for dimension 2. This gives hope that more efficient heaviness testers in higher dimensions can also be found.

2 Preliminaries

Notation. We use the notation $x \in_R S$ to indicate x is chosen uniformly and 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 $i \in \{1, \ldots, d\}$. Consider a function

$f : \mathcal{D}^d \to \mathcal{R}$, where $\mathcal{D}^d$ is a d-dimensional lattice. If $x, y \in \mathcal{D}^d$ are such that $x \prec y$ then if $f(x) \leq f(y)$ we say that x and y are in *monotone order*. We say f is *monotone* if for all $x, y \in \mathcal{D}^d$ such that $x \prec y$, x and y are in monotone order.

Approximate PCP. The approximate PCP model is introduced in [7]. The verifier has access to a written proof, Π, which it can query in order to determine whether the theorem it is proving is *close* to correct. More specifically, if on input x, the prover claims $f(x) = y$, then the verifier wants to know if y is close to $f(x)$.

Definition 1. *[7] Let $\Delta(\cdot, \cdot)$ be a distance function. A function f is said to have a* $t(\epsilon, n)$-approximate probabilistically checkable proof system *with distance function Δ if there is a randomized verifier $\mathcal{V}$ 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 $\Delta(y, f(x)) = 0$, there is a proof, Π, such that $\mathcal{V}^{\Pi}$ 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 proofs Π', $\mathcal{V}^{\Pi'}$ 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 probabilistically checkable proof protocol can be repeated $O(\lg 1/\delta)$ times to get confidence $\geq 1 - \delta$. We occasionally describe the verifier's protocol as an interaction with a prover. In this interpretation, it is assumed that the prover is bound by a function which is fixed before the protocol begins. It is known that this model is equivalent to the PCP model [8]. The verifier is a RAM machine which can read a word in one step.

We refer to PCP using the distance function $\Delta(y, f(x)) = \max\{0, 1 - f(x)/y\}$as an *approximate lower bound* PCP : if $f(x) \geq y$ then Π causes $\mathcal{V}^{\Pi}$ to pass; if $f(x) < (1 - \epsilon)y$ then no proof Π' convinces $\mathcal{V}^{\Pi'}$ with high probability. This distance function applied to our bin-packing protocol will show that if a prover 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 [7] that the protocols we give can be employed to prove the existence of suboptimal solutions. In particular, if the prover knows a solution of value v, it can prove the existence of a solution of value at least $(1 - \epsilon)v$. Since v is not necessarily 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 prover may not have computational powers outside of deterministic polynomial time, but might employ very good heuristics. In addition, since the prover is much more powerful than V it may use its computational abilities to get surprisingly good, yet not necessarily optimal, solutions.

Heaviness Testing. Our methods all rely on the ability to define an appropriate *heaviness* property of a function f. In the multidimensional case, heaviness is defined so that testing if a domain element is heavy can be done very efficiently in the size of the domain, and such that all heavy elements in the domain which are comparable according to $\prec$ are in monotone order.

We give a simple motivating example of a heaviness test for $d = 1$ from [6]. This one-dimensional problem can be viewed as the problem of testing whether a list $L = (f(1), f(2), \ldots, f(n))$ is mostly sorted. Here we assume that the list contains distinct elements (a similar test covers the nondistinct case). Consider the following for testing whether such a list L is mostly sorted: pick a point $x \in L$ uniformly and at random. Perform a binary search on L for the value x. If the search finds x then we call x *heavy*. It is simple to see that if two points x and y are heavy according to this definition, then they are in correct sorted order (since they are each comparable to their common ancestor in the search tree). The definition of a heaviness property is generalized in this paper. We can call a property a *heaviness property* if it implies that points with that property are in monotone order.

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

1. $\forall x < y$ $H(x) \wedge H(y)$ *implies* $f(x) \leq f(y)$
2. *In a monotone list all points have property H*

If a point has a heaviness property H then we say that point is *heavy*. There may be many properties which can be tested of points of a domain which are valid heaviness properties. A challenge of designing heaviness tests 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 $\mathcal{D}_{\prec} = [1, \ldots, n]^d$ be a domain, and let $f : \mathcal{D} \to \mathcal{R}$ be a function on D. Let $S(\cdot, \cdot)$ be a randomized decision procedure on $\mathcal{D}$. Given security parameter δ, we will say S is a* heaviness test *for x if*

1. *If for all $x \prec y$, $f(x) \leq f(y)$ then* $S(x, \delta) = \mathrm{Pass}$
2. *If x is not heavy then* $\Pr(S(x, \delta) = \mathrm{Fail}) > 1 - \delta$

The heaviness tests we consider enforce, among other properties, local multidimensional monotonicity of certain functions computed by the prover. It turns out that multidimensional heaviness testing is more involved that 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 heaviness tests exist. Such tests would immediately improve the efficiency of our approximate PCP verifier for bin-packing.

Permutation Enforcement. Suppose the values of a function f are given for inputs in $[n]$ in the form of a list $y_1, \ldots, y_n$. Suppose further that the prover would like to convince the verifier that the y_i's are distinct, or at least that there are $(1 - \epsilon)n$ distinct y_i's. In [7], the following method is suggested: The prover writes 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 $A(f(i)) = i$. 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 are honest suffices to ensure that there are at least $(1 - \epsilon)n$ distinct y_i's in $O(1/\epsilon)$ time. We refer to A as the *permutation enforcer*.

3 Multidimensional Bin-Packing

We consider the d-dimensional bin-packing problem. We assume the objects to be packed are d-dimensional rectangular prisms, which we will hereafter 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 $x \prec y \prec x + w$ 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 $\mathcal{D} = (N_1, \ldots, N_d)$.

It turns out to be convenient to cast our problem as a maximization problem. We define the d-dimensional bin-packing problem as follows: given n blocks, the dimensions of a bin, and an integer k, find 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 we take $N = \max_i N_i$. In all of these protocols, we assume that the block and bin dimensions fit in a word.

In this protocol, we assume that the prover is trying to convince the verifier that all the blocks can be packed in k bins. We address the more general version of this problem in the full version of this paper. In doing so we use the approximate lower bound protocol for set size from [7].

We require that the prover 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 the bins used by the prover, the verifier accepts. If 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 either accepts or rejects.

3.1 A First Representation of a Packing

We represent a bin as a d-dimensional grid with the appropriate length in each dimension. The prover 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 can give requirements by which the prover can define a monotone function on the grid using these labels only if he knows a feasible packing. To show such a reduction exists, we first define a relation on blocks.

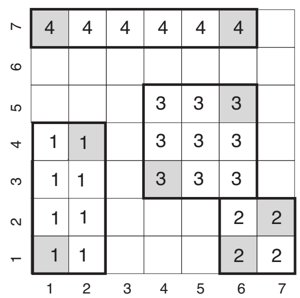

Fig. 1. A 2D Encoding

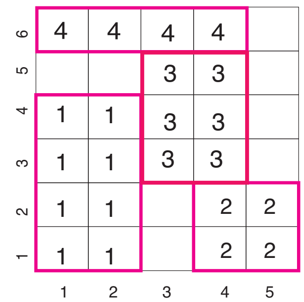

Fig. 2. Compressed Grid Encoding

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 1 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: 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 blocks is feasible. Given such an ordering, it is easy to construct a monotone function.

Lemma 1. *Given a feasible packing of a bin with blocks, we can label the blocks with distinct integers such that when we assign each grid element in the d-dimensional grid (of the bin) with the label of the block occupying it, we get a monotone function on this grid.*

Proof. Without loss of generality, assume that the bin is filled up completely. We know that by inserting extra "whitespace" blocks we can fill up the bin. It can be shown that the bin can be packed in such a way that $4n$ whitespace blocks are sufficient. 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 function on the grid.

Now we can describe the proof that the prover will write down. The proof will consist of three parts: the first one is a table which will have an entry for each block containing the label assigned to the block; a pointer to the bin where the object was assigned and the locations of the two (lowest and highest) corners of the block in this bin. The second part is a permutation enforcer on the blocks and the labels of the blocks.

Finally, the third part consists of a d-dimensional grid of size $\prod[N_j]$ for each bin used that numbers each grid element with the label of the block occupying it.

3.2 Testing Multidimensional Bin-Packing Using Heaviness

The heaviness test we have defined can be used to test that the prover's labeling agrees with a monotone function. By using Observation 1, we will be able to show 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" should have the properties that (1) a good block is actually packed inside a bin, and it is not overlapping any other "good" block; and (2) we can efficiently test a block for being good. Then, the verifier will use sampling to ensure that at least $1 - \epsilon$ fraction of the blocks are "good" in the protocol.

Definition 6. *The block i with dimensions $w = (w_1, \ldots, w_d)$ is* good *with respect to an encoding of a packing if it has the following properties:*

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

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

Lemma 2. *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. Without loss of generality, b_1 precedes b_2. Since these blocks overlap, the lowest corner of b_2, $l^{(b_2)}$, is smaller than the highest corner of b_1, $h^{(b_1)}$ ($l^{(b_2)} \prec h^{(b_1)}$). We know, by definition of a heaviness tester, that two comparable heavy points on the grid do not violate monotonicity. But, 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 1. *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: The verifier chooses a block randomly from the input, and using the encoding described above, confirms that the block is good. Testing a block for being good involves $O(d)$ comparisons for the first two conditions in the definition, O(1) time for checking the unique labeling using the permutation enforcer, and 2 heaviness tests for the third condition. The verifier repeats this $O(1/\epsilon)$ times to ensure at least $(1 - \epsilon)$ fraction of the blocks are good.

Theorem 1. *There is an $O((1/\epsilon)T(N, d))$-approximate lower bound PCP for the d-dimensional bin packing problem where $T(N, d)$ is the running time for a heaviness tester on $\mathcal{D} = [N_1] \times \cdots \times [N_d]$.*

3.3 A Compressed Representation of a Packing

The previous protocol requires the prover to write down a proof whose size depends on the *dimensions* of the bins to be filled, since the values N_i were based on the actual size of the bins given. We show here how the prover may write a proof which depends only on the number, n, of objects to be packed. In the protocol from the previous section the verifier calls the heaviness tester only on grid elements which correspond to the lowest or the highest corners of the blocks. We use this observation for a compressed proof.

The prover constructs a set of *distinguished coordinate* values S_k for each dimension $k = 1, \ldots, d$. Each set is initially empty. The prover considers each block i and does the following: for the lower corner, $l^{(i)} = (c_1, \ldots, c_d)$, and higher corner, $h^{(i)} = (e_1, \ldots, e_d)$, of block i, the prover computes $S_i \leftarrow S_i \cup \{c_i\} \cup \{e_i\}$. After all the blocks are processed, $|S_i| \leq 2n$. The *compressed grid* will be a sublattice of $\mathcal{D}$ with each dimension restricted to these distinguished coordinates, that is the set $\{\langle x_1, \ldots, x_d \rangle | x_i \in S_i\}$. This grid will contain in particular all the corners of all the blocks and the size of this proof will be at most $O((2n)^d)$. Note that although in the previous test we have added "whitespace" blocks to generate our monotone numbering, those blocks themselves were never tested, hence they do not affect the number of distinguished coordinates. 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 prover's compression is valid.

The proof consists of four parts. First the prover implicitly defines the proof from the previous section, which we refer to as the *original grid*. The prover then writes down a table containing the *compressed grid*. In each axis, the prover labels the coordinates $[1, \ldots, 2n]$ and provides a *lookup-table* (of length $2n$) for each axis which maps compressed grid coordinates to original grid coordinates. Finally the prover writes down the list of objects with pointers to the compressed grid, and a permutation enforcer as before. In Figure 2 , we give the compressed encoding of the packing from Figure 1.

Protocol. By making the prover write only a portion of the proof from the first protocol, we provide more opportunities for the prover to cheat. For example, even if the prover uses the correct set of hyperplanes for the compression, he 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 6 in the compressed grid[1]. 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.

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

Correctness. Any two good objects do not overlap in the compressed grid, by applying Lemma 2. Furthermore, since the labels of good objects in the lookup table are heavy, it follows that two good objects do not overlap in the original grid either. Certainly, since the corresponding values in the lookup table form a monotone sequence, the prover 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 2. *There is an $O((1/\epsilon)T(n,d))$-approximate lower bound PCP for the d-dimensional bin packing problem with proof size $O((2n)^d)$, where $T(n,d)$ is the running time for a heaviness tester on $\mathcal{D} = [n]^d$.*

3.4 Further Applications

Multidimensional Routing A graph G with edge-capacity constraints is given along with a set of desired messages which are to be routed between vertex pairs. Each message has a bandwidth requirement and a duration. If $\mathcal{P}$ knows how to route f of these messages, he can convince $\mathcal{V}$ that a routing of $\geq (1-\epsilon)f$ exists. We sketch the method: The prover presents the solution as a 2D bin packing proof, with one bin for each edge: one dimension corresponds to the bandwidth, the other to the duration. The portion of a message routed along a particular bin is a 2D block. To verify that a routing is legal, $\mathcal{V}$ selects a message at random and the prover provides the route used as a list of edges. The verifier checks that sufficient bandwidth is allocated and that durations are consistent along all edges of the route and that the message *(block)* is "good" with respect to the packings of blocks in each of the edges *(bins)*. If we assume that the maximum length of any routing provided by the prover is length k, this yields a protocol with running time $O((k/\epsilon) \cdot \log^2(n))$, where n is the maximum number of calls ever routed over an edge. To achieve this running time we employ the heaviness tester in Section 4. Higher dimensional analogues of this problem can be verified by an extension of these methods.

Recursive Bin Packing At the simplest level the recursive bin packing problem takes as input a set of objects, a list of container sizes (of unlimited quantity), and a set of bins. Instead of placing the objects directly in the bins, an object must first be fit into a container (along with other objects) and the containers then packed in the bin. The goal is to minimize the total number of bins required for the packing. We can solve this problem by applying an extension of our multidimensional bin-packing tester. In particular, we define an object as *good* if it passes the goodness test (with respect to its container) given in Section 2 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 objects are good and hence that $(1-\epsilon)$ fraction of the objects can be feasibly packed. For a k-level instance of recursive bin packing, therefore, the prover will write k compressed proofs and $O(k/\epsilon)$ goodness tests will be needed.

3.5 Can Monotonicity Testing Help?

Given the conceptual 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 objects in a $(2n)^d$ sized domain. If a monotonicity tester can determine that the domain of our compressed proof is 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 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 Heaviness Tests

We give two heaviness tests for functions on a domain isomorphic to an integer lattice. The domains are given as $\mathcal{D} = [1,\ldots,n]^d$. The range can be any partial order, but here we use $\mathbb{R}$, reals. Both tests which follow determine that a point is heavy in $O((2\log n)^d)$ time, yielding efficient bin packing tests for small values of d. In particular, the examples applications of bin packing which we have cited typically have dimension less than 3. For complete proofs, please consult the full version of the paper [2].

4.1 The First Algorithm

We extend the protocol of [6] to multidimensional arrays. On input x our test compares x to several random elements y selected from a set of carefully chosen neighborhoods around x. It is tested that x is in order with a large fraction of points in each of these neighborhoods. From this we can conclude that any two comparable heavy points a and b can be ordered by a mutually comparable point c such that $a < c < b$ and $f(a) < f(c) < f(b)$. The test is shown in Figure 4.

Proof of Correctness We consider a set of $\log^d n$ carefully chosen neighborhoods around a point x. We say that x is *heavy* if for a large fraction of points y in each of these neighborhoods, $f(x)$ and $f(y)$ are monotonically ordered. We are able to show from this that for any two heavy points x and y, two of these regions can be found whose intersection contains a point z with the property that $x < z < y$ and $f(x) < f(z) < f(y)$. Hence this defines a valid heaviness property. The efficiency of the test is bound by the fraction of points in each neighborhood which must be tested, which is given to us by Chernoff bounds. It follows that

Theorem 3. *Algorithm* `Heavy-Test` *is a heaviness tester performing* $O(\log(1/\delta)(2\log(n))^d)$ *queries.*

4.2 The Second Algorithm

This algorithm is based on a recursive definition of heaviness. 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 dimension d hypercube, C , consider a subdividing operation ϕ which maps C into 2^d congruent subcubes. This operation passes through the center d hyperplanes parallel to each of the axes of the hypercube. This is a basic function in our algorithm. For notational convenience, we extend ϕ to Φ which acts on sets of cubes such that $\Phi(\{x_1,\ldots,x_n\}) = \{\phi(x_1),\ldots,\phi(x_n)\}$. It is now possible to compose Φ with itself. We also define a function $\hat{\Phi}(x,C) = S \Rightarrow x \in S \in \Phi(C)$. This function is also a notational convenience which identifies the subcube a point lies in after such a division.

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 $\varrho : C^2 \rightarrow \mathcal{Z}$ such that $\varrho(x,y) = r \Rightarrow \hat{\Phi}^r(x,C) = \hat{\Phi}^r(y,C)$ and $\hat{\Phi}^{r+1}(x,C) \neq \hat{\Phi}^{r+1}(y,C)$. That is, the $r+1$st composition of Φ on C separates x from y.

Definition 7. *A point x is heavy in a domain $\mathcal{D} = [n]^d$ if the $2d$ perpendicular projections of x onto each cube in the series $\hat{\Phi}(x,\mathcal{D}),\ldots,\hat{\Phi}^{\log n}(x,\mathcal{D})$ of shrinking cubes are all heavy in dimension $d-1$. The domains $\mathcal{D}'$ for these recurive tests are the respective faces of the cubes. When $d = 1$ we use the test of* [6].

We can now give the 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 $\{s_1,\ldots,s_k\} = \{\hat{\Phi}^1(x,C),\ldots,\hat{\Phi}^k(x,C)\}$ where $k = \lceil \log(n) \rceil$. Note that $s_k = x$. At each cube s_k perform the following test: (1) Compute the $2d$ perpendicular projections $\{p_1,\ldots,p_{2d}\}$ of x onto the $2d$ faces of s_k. (2) Verify that f is consistent with a monotone function on each of the $2d$ pairs (x,p_k). (3) If $d > 1$ recursively test that each of the points p_i is heavy over the reduced domain of its corresponding face on s_k. If $d = 1$, we use the heaviness test of [6]. This test is shown in Figure 3.

Theorem 4. *If x and y are heavy and $x < y$ then $f(x) < f(y)$*

Proof. (by induction on d). Let $r = \varrho(x,y)$. Let $S = \hat{\Phi}^r(x,C)$. Let $s_x = \hat{\Phi}^{r+1}(x,C)$ and $s_y = \hat{\Phi}^{r+1}(y,C)$. There is at least one plane perpendicular to a coordinate axes passing through the center of S which separates x and y. This plane also defines a face of s_x and of s_y, which we denote as f_x and f_y respectively. By induction we know the projections of x and y onto these faces are heavy. Since y dominates x in every coordinate, we know that $p_x < p_y$. Inductively we can conclude from the heaviness of the projection points that $f(p_x) < f(p_y)$. Since we have previously tested that $f(x) < f(p_x)$ and $f(p_y) < f(y)$ we conclude $f(x) < f(y)$.

Running time analysis If we let $H_d(n)$ be the number of queries made by our algorithm in testing that a point of the function $f : \mathcal{Z}_n^d \rightarrow \mathcal{S}$ is heavy, then we can show

Lemma 3. *For all $d > 1$, for sufficiently large n, $H_d(n) \leq (d-1)\log^d(n)\log(1/\delta)$*

Proof. By induction. For the case $d = 1$ we employ the spot checker algorithm from [6], which performs $\log(1/\delta)\log(n)$ queries to determine that a point is heavy.

```
RecursiveTest(f,ε,δ,D,d)
if d = 1
  δ' ← δ/(d! log^d n)
  return SpotCheckTest(f,ε,δ')
else
  for i = 1...log n
    Φ = Φ̂^i(x,C)
    {p_1,...,p_d} = projections
       of x onto Φ
    for k = 1...d
      C ← the face of Φ̂^i(x,D)
         containing p_k
      HeavyTest(f,ε,δ,D,d)
    end
  end
end
return PASS
```

Fig. 3. Algorithm RecursiveTest

```
Heavy-Test(f,x,ε,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 ≤ n
   h ← (h_1,...,h_d)
   if  (f(x) > f(x + h))  return FAIL
return PASS
```

Fig. 4. Algorithm Heavy-Test

Theorem 5. *Algorithm* `RecursiveTest` *is a heaviness tester performing* $O((d\log(d)+d\log\log(n)+log(1/\delta))(d-1)\log^d(n))$ *queries.*

Proof. The confidence parameter $\delta' = \delta/(d!\log^d(n))$ which appears in Figure 3 arises because the probability of error accumulates at each recursive call. Now apply Lemma 3.

References

1. 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.
2. T. Batu,R. Rubinfeld,P. White. Fast approximate PCPs for multidimensional bin-packing problems. *http://simon.cs.cornell.edu/home/ronitt/PAP/bin.ps*
3. L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols, *Computational Complexity*, pp. 3–40, 1991.
4. L. Babai, L. Fortnow, C. Lund, and M. Szegedy. Checking computations in polylogarithmic time. *Proc. 31st Foundations of Computer Science*, pp. 16–25, 1990.
5. Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron and A. Samorodnitsky Improved Testing Algorithms for Monotonicity. RANDOM '99.
6. F. Ergun, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *Proc. 30th Symposium on Theory of Computing*, pp. 259–268, 1998.
7. F. Ergün, R. Kumar, R. Rubinfeld. Fast PCPs for approximations. *Proc. 31st Symposium on Theory of Computing*, 1999.
8. L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545-557, 1994.
9. O. Goldreich,S. Goldwasser, E. Lehman, D. Ron. Testing Monotonicity *Proc. 39th Symposium on Foundations of Computer Science*, 1998.