Efficient Approximation Algorithms for Tiling and Packing Problems With Rectangles*

Piotr Berman† Bhaskar DasGupta‡ S. Muthukrishnan§ Suneeta Ramaswami¶

July 11, 2001

Abstract

We provide improved approximation algorithms for several rectangle tiling and packing problems (RTILE, DRTILE and d-RPACK) studied in the literature. Most of our algorithms are highly efficient since their running times are near-linear in the sparse input size rather than in the domain size. In addition, we improve the best known approximation ratios.

1 Introduction

In this paper, we study several rectangle tiling and packing problems. These are natural combinatorial problems that arise in many applications in databases, parallel computing and image processing. We present new approximation algorithms for these problems. In contrast to the previously known best results, we meet a crucial demand of most of these applications, namely, most of our algorithms work on sparse inputs and/or in high dimensions more efficiently. In addition, our algorithms have better approximation bounds than the previously known algorithms. Furthermore, the algorithms are simple to implement. In what follows, we will first formally define the problems before presenting our results.

1.1 The Rectangle Tiling and Packing Problems

We study the two classes of problems as described below. The following definitions and notations are used throughout the rest of the paper.

- For the RTILE/DRTILE problems (as described below), we will always use bold letters to denote arrays/rectangles, and respective regular letters to denote their weights. In particular, input array $\mathbf{A}$ has weight A, and $\mathbf{R}_i$, the i^{th} row of of a two-dimensional array $\mathbf{A}$, has weight R_i.

*A preliminary version of this paper appeared in 12th ACM-SIAM Symposium on Discrete Algorithms, January 2001, pp. 427-436.

†Department of Computer Science, Pennsylvania State University, University Park, PA 16802. Email: berman@cse.psu.edu. Supported in part by NSF grant CCR-9700053 and by NLM grant LM05110.

‡Department of Computer Science, University of Illinois at Chicago, 851 South Morgan Street (SEO Building, M/C 152), Chicago, IL 60607-7053. Email: dasgupta@eecs.uic.edu. Supported in part by NSF Grant CCR-9800086.

§AT& T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932. Email: muthu@research.att.com.

¶Department of Computer Science, Rutgers University, Camden, NJ 08102. Email: rsuneeta@crab.rutgers.edu.

- Given an array $\mathbf{A}$ a *tile* of $\mathbf{A}$ is any rectangular subarray of $\mathbf{A}$, and a *partition* of $\mathbf{A}$ into tiles is one in which each element of the array falls in precisely one tile of $\mathbf{A}$ with the tiles being disjoint (i.e., sharing no elements).
- An array $\mathbf{A}$ is called a $\{0,1\}$-array if all of its entries are either 0 or 1; otherwise it is called a *general* or *arbitrary* array.
- All logarithms in this paper are in base 2 if the base is not mentioned explicitly.

With the above notations and definitions, we define the two classes of problems that are of interest in this paper.

RTILE/DRTILE problems. Given a two dimensional array $\mathbf{A}$ of size $n \times n$ containing non-negative integers[1], partition $\mathbf{A}$ into at most p rectangular tiles so that the maximum weight of any tile is minimized, where the *weight* of a tile is the sum of all elements of $\mathbf{A}$ that fall in it. Our emphasis is on the *sparse* version of the problem, that is, one in which the total number of nonzero entries in $\mathbf{A}$ is at most m which is likely to be smaller than n^2, the total number of entries in $\mathbf{A}$.

A natural variant of RTILE is its dual, referred to as the DRTILE problem, in which we are given a weight bound w and required to minimize the number of tiles needed to partition the given array so that *no* tile has weight larger than w. Another natural variant is the straightforward extension of the RTILE/DRTILE problems to d-dimensions, where $\mathbf{A}$ is a d-dimensional array of size $n \times n \times \cdots \times n$ and d-dimensional hyper-rectangles are considered in the tiling[2].

d-RPACK problem. Given a set of p weighted d-dimensional, axis-parallel hyper-rectangles with endpoints in $\{1, 2, 3, \ldots, n\}$ and a parameter k, find a collection of at most k disjoint hyper-rectangles of maximum total weight[3]. The set of all possible hyper-rectangles in d dimensions is $n^{\Omega(d)}$, but p is likely to be significantly smaller, that is, the sparse case is of practical interest.

The d-RPACK problem on a set of hyper-rectangles is equivalent to finding the maximum-weight independent set in the intersection graph of these hyper-rectangles, where there is an edge between two nodes representing two distinct hyper-rectangles if and only if they intersect (this problem has been studied with an alternate definition of the interference graph in [BNR96]). The dual of the RPACK problem is to find the minimum cardinality subset of disjoint hyper-rectangles with total weight at least w, for some given weight-bound w. Since it is NP-hard to even find a feasible solution for this dual problem [FPT81], it cannot be approximated to within any factor. Hence, we do not consider this dual problem any further.

[1] All our results will generalize to $n_1 \times n_2$ array. Henceforth, when we refer to *arbitrary array* $\mathbf{A}$ we imply one with non-negative integers. While our results do extend to the case when the elements are positive reals, we focus on the non-negative integer case motivated by the applications.

[2] Obviously, every element of the given array has to be at most w for a solution to the DRTILE problem to exist

[3] All our results generalize to the case when the endpoints of the i^{th} dimension of the hyper-rectangle lies in $\{1, 2, \ldots, n_i\}$, for some positive integers $n_1, n_2, \ldots, n_d$.

1.2 Motivating Applications

Rectangle tiling and packing problems as described in the previous section are natural combinatorial problems arising in many scenarios. For motivation, we will very briefly review one example application scenario for each and leave others to be found in the references.

Equisum Histograms in Databases. Databases such as DB2, SQLServer, Oracle, Sybase, Ingres etc. routinely keep histograms to optimize query execution [K80]. These histograms attempt to capture the distribution of attribute values in the database, for example, the joint distribution of the frequency of attribute values (e.g., age, salary, savings, etc.); the goal is to do so using limited memory and reliable accuracy. While the study of histograms is fundamental in areas such as Statistics, the use of multidimensional histograms in databases was initiated in [MD88], and has been intensely researched since then (see [P97] for a survey).

The RTILE problem is precisely that of constructing what is known as the Equisum histogram on 2 (more generally d) numerical attributes as defined in [MD88]. In this context, $\mathbf{A}$ is defined as follows: $\mathbf{A}[i,j]$ is 1 if there exist database records with attributes values i and j respectively (e.g., salary of 100K and savings of 500K, respectively) and 0 otherwise; p, the number of tiles, represents the space allocated for the histogram; and w, the maximum weight of a tile, affects the accuracy of estimating the joint distribution. This gives the RTILE problem where A is a $\{0,1\}$-array. More generally, we may let $\mathbf{A}[i,j]$ be the number of database records that have attribute value i and j respectively, and we would obtain the RTILE problem with non-negative integral elements. Both versions of the problem are of interest in selectivity estimation. (DRTILE problem is the dual of RTILE, which is equally interesting for building histograms.)

An important feature of this application is the sparsity of the input. That is, $\mathbf{A}[i,j]$ is nonzero only for the combination of attribute values that are present in the database, which is typically significantly smaller than the set of all possible combinations of attribute values. It is imperative to exploit this sparseness in database applications.

RTILE and DRTILE problems, and their variants, are also of interest in other applications in load balancing, database compression, data partitioning for parallel computers, image and video processing, etc. See discussion and references in [MPS99, KMP98].

Database Decision Support. Database mining systems generate association rules — rules of the form $C_1 \rightarrow C_2$, that is, if condition C_1 is satisfied in the database, C_2 follows. Correlation rules of this form need support and confidence to be tagged by database mining systems. For numeric attributes, the conditions on the left take the form of *clusters* (ranges), an anecdotal example being: $(Age \in [25,45]) \cap (Balance \in [15K - 40K]) \rightarrow (CarLoan = Yes)$. (Here, there are two numeric attributes, namely, Age and Balance. In general, there could be many other numeric attributes such as Mutual Fund Investments etc.) Study of such clustered association rules can be found in [FM+96a, FM+96b, RS99]. Database mining systems can now generate the set of all such rules for a given database, given thresholds on confidence and support, and tag each with a weight that shows their gain or value. Following that, database decision systems choose a subset of these rules for further development, such as marketing. A common formulation of this task is to choose k disjoint rules of largest total gain or value (for example, see [FM+96a, RS99] for formulation of this problem and extensive experimental study). This is precisely the d-RPACK problem where the d dimensions correspond to the numeric attributes.

Besides the above application, the d-RPACK problem arises in various resource allocation problems. It is also a natural combinatorial problem when viewed as the maximum-weight independent set problem on the corresponding intersection graph of the hyper-rectangles. In the above motivating application, the set of hyper-rectangles that is generated is typically much smaller than the set of all possible hyper-rectangles. *The focus is therefore again on the sparse input case.*

1.3 Summary of Our Results and Related Research

Since the RTILE, DRTILE and d-RPACK problems are known to be NP-Hard [FPT81, KMP98] in two or more dimensions[4], our goal is to design approximation algorithms with guaranteed performance bounds. Naturally, our focus is to design approximation algorithms with better performance ratios than previously known, but additionally, our focus is to design such approximation algorithms whose time complexity is efficient as a function of the *sparse input size* (e.g., $m+n$ in the RTILE and DRTILE problems in two dimensions, p in the d-RPACK problem) rather than merely being efficient in the universe size (i.e., n^2 and n^d, respectively). None of the existing algorithms for these problems fully meet the latter goal. Table 1 summarizes our main results and compares them with the previously best known results.

Problem	Our results		Previous best		
	ratio	time-O	ratio	time-O	reference
RTILE, **A** is $\{0,1\}$	2	$n+m$	$9/4$	$n^2+p\log n$	[KMP98]
RTILE	$11/5$	$n+m$	$7/3$	n^2	[S99]
DRTILE, **A** is $\{0,1\}$	2	$n+m$	$9/4$ 2	$n^2+p\log n$ n^{10}	[KMP98] [KMP98]
DRTILE, d-dim	$2d-1$	$d(m+n)$	$2d$	$n^d+pd2^d\log n$	[S99, SS99]
d-RPACK	$(\lfloor 1+\log n\rfloor)^{d-1}$	$dp\log^{\varepsilon} p$ $+dn\frac{\log n}{\log\log n}+pk$	$6\lfloor 1+\log n\rfloor$	$n^2p+np^2\log n$	[KMP98] $d=2$ only
	$\left(\lfloor 1+\frac{\log n}{c}\rfloor\right)^{d-1}$	$p^{(2^c-1)^{d-1}+1}\,dk$			

Table 1: Summary of our main results and comparisons with previous results. For the RTILE/DRTILE problems, n is the size of each dimension, m is the number of non-zero entries in the array ($m \le n^d$ for a d-dimensional array), and $p = A/w$. Unless otherwise noted **A** has arbitrary entries. For the d-RPACK problem, p is the total number of hyper-rectangles, n is the size of each dimension, k is the number of rectangles to be selected, and $\varepsilon > 0$ and $c \ge 1$ are any two constants.

We specially note the following points:

- *Tiling Problems:* Our algorithms for the RTILE/DRTILE problems take time linear in the sparse

[4]The one dimensional versions of RTILE, DRTILE and d-RPACK problems can be solved in polynomial time by dynamic programming and greedy approaches [KMP98].

input size, that is $O(m+n)$, and are simple to implement.

- *Packing Problems.* The $(\lfloor 1+\log n \rfloor)^{d-1}$-approximation algorithm for packing problem takes time almost linear in sparse input size p. No results were previously known for the d dimensional case; straightforward extension of the known result for the two dimensional case [KMP98] would be an $\Omega(\log^{2d-1} n)$ approximation algorithm taking $n^{\Omega(d)}$ time. Hence, our result is a substantial improvement. We can further improve the approximation ratio of our algorithm at the expense of increasing the running time as shown in the table above; the algorithm is now more complicated but it is of theoretical interest since it follows that the logarithmic approximation barrier, that is common to many natural combinatorial problems such as the set cover problem, does not hold for this problem.

1.4 Brief Technical Overview of Results

Tiling Problems. While previous algorithms used the concept of "thick cuts" or "medium cuts" and adopted a divide-and-conquer approach, we adopt a "sweep" based technique and develop the concept of "good rectangles" in the array, i.e., those rectangles whose total weight is at least g (for some parameter g carefully chosen to justify the approximation bounds) and combine these rectangles, whenever necessary, in a local manner. The benefit is that while thick or medium cuts are somewhat expensive to find, good rectangles can be found quite simply by sweeping through the input and combining them is also algorithmically easy. Thus our algorithms have efficient implementation in the sparse input size as applications demand. Somewhat surprisingly, such simple approaches in fact yield improved approximation ratios.

The improvement in the approximation ratio for the DRTILE problem in d dimensions over the previously best known algorithm uses an alternative lower bound in addition to the usual lower bound considered in [KMP98, S99, SS99]. The two lower bounds together lead to an improved analysis of the performance ratio.

Packing problem. The $(\lfloor \log(n+1) \rfloor)^{d-1}$-approximation algorithm that we present uses a divide-and-conquer technique. The exact solution at each level is found by "translating" the rectangles in the subproblems at each node in the level so that their "lower dimensional projections" do not interfere. For the improved result with approximation ratio $\left(\left\lfloor 1+\frac{\log n}{c}\right\rfloor\right)^{d-1}$ we do this at several consecutive levels of the divide-and-conquer simultaneously. As a result, the algorithm becomes more involved.

1.5 The Map

In Section 2, we present some necessary definitions. Our approximation algorithms for the RTILE and DRTILE problems are in Section 3. Section 4 contains our results on d-RPACK problem. Concluding remarks are in Section 5.

2 Definitions and Preliminaries

Unless otherwise stated, all rectangles are *axis-parallel* and all weights are *non-negative*. A *d-dimensional hyper-rectangle* is $[x_{11}, x_{12}] \times [x_{21}, x_{22}] \times \cdots \times [x_{d1}, x_{d2}] = \times_{i=1}^{d} [x_{i1}, x_{i2}]$, where each $[x_{i1}, x_{i2}]$ is an interval on the real line and $\times$ denotes the Cartesian product. A 2-dimensional hyper-rectangle is simply called a *rectangle*. Two d-dimensional hyper-rectangles $\times_{i=1}^{d} [x_{i1}, x_{i2}]$ and $\times_{i=1}^{d} [y_{i1}, y_{i2}]$ intersect if and only if $[x_{i1}, x_{i2}] \cap [y_{i1}, y_{i2}] \neq \emptyset$ for every i. A *partition* of an array $\mathbf{A}$ is a collection of (axis-parallel) mutually disjoint rectangles (also referred to as *tiles*) that cover $\mathbf{A}$. The *weight* of a rectangle in a partition of $\mathbf{A}$ is the sum of all array elements covered by the rectangle.

If a given $n \times n$ array is sparse, containing m non-zero entries ($n \leq m \leq n^2$), then it can be efficiently represented in $O(m+n)$ space using the standard representation as an array of row lists; the list of the i^{th} row contains an entry of the form (j, x) for every positive array entry $\mathbf{A}[i, j] = x$ and the row lists are sorted by the column numbers of the entries. A similar standard sparse representation can be used for a d-dimensional array $\mathbf{A}$: a non-zero entry $\mathbf{A}[i_1, i_2, \ldots, i_d]$ is represented by a $(d+1)$-tuple $(i_1, i_2, \ldots, i_d \ ; \ \mathbf{A}[i_1, i_2, \ldots, i_d])$ and the tuples are lexicographically sorted in their first d coordinates[5]. Hence, we assume that our input sparse arrays are represented this way.

3 Approximating the Tiling Problem

First, we describe a general slicing technique that is used by the algorithms in Theorem 1 and Theorem 2 for the RTILE and DRTILE problems (Theorem 3 uses a slightly different slicing technique for the DRTILE problem in d dimensions). Next we focus on the RTILE and DRTILE problems on $\{0,1\}$-arrays and later consider these problems on general arrays.

3.1 Slicing Arrays

Definition 1 *A tile (rectangle) is defined to be g-good provided it has weight at most g.*

The goal of the slicing algorithm is to partition the input array $\mathbf{A}$ into slices (rectangles) depending on the parameter g. Selection of g will depend on the particular algorithm and the particular problem. Algorithms using this slicing technique have a common beginning: first, the value of g is established, and then the given array $\mathbf{A}$ is partitioned into slices depending on g. The general scheme for the slicing algorithm is shown below. For $j < l$ the rows from $\mathbf{R}_{t[j-1]+1}$ to $\mathbf{R}_{t[j]}$ form the j^{th} slice, denoted by $\mathbf{S}_j$. The rows with numbers larger than $t[l-1]$ form the *remainder* slice, $\mathbf{S}_l$. For $j < l$, we define the *top* of the slice $\mathbf{S}_j$ as $\mathbf{T}_j = \mathbf{R}_{t_j}$. The other rows of this slice form its *base*, $\mathbf{B}_j$.

$t[0] \leftarrow 0$
$j \leftarrow 1$
$slice\_weight \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**

[5] If the m non-zero entries of a d dimensional array $\mathbf{A}$ are given in some arbitrary order, then they can be lexicographically sorted using bucket sort in $O(d(m+n))$ time. This will not increase the overall time complexity of our algorithms

```
        slice_weight ← slice_weight + R_i
        if slice_weight > g then
            t[j] ← i
            j ← j + 1
            slice_weight ← 0
    l ← j
```

It is easy to see that we can partition $\mathbf{A}$ into slices and compute all R_i's, S_j's, T_j's and B_j's easily in $O(n+m)$ time.

3.2 The RTILE and DRTILE Problems on $\{0,1\}$-arrays

Lemma 1 *If the input array* $\mathbf{A}$ *is a* $\{0,1\}$*-array, we can partition it into* $\lceil 2A/g \rceil$ g*-good rectangles in* $O(n+m)$ *time.*

Proof. Note that the remainder slice is always g-good and the other slices are not. To prove the bound on the number of g-good rectangles, it suffices, for $j < l$, to divide slice $\mathbf{S}_j$ into $a_j < 2\frac{S_j}{g}$ g-good rectangles. We consider three cases.

Case 1: $T_j \leq g$. In this case both $\mathbf{T}_j$ and $\mathbf{B}_j$, the top and the base of $\mathbf{S}_j$, are g-good; so they constitute our partition of $\mathbf{S}_j$ into rectangles. Since $S_j > g$, we have $a_j = 2 < 2S_j/g$.

Case 2: $T_j > g$ and $2S_j/g \leq 3$. This implies that $B_j < g/2$. Assume that the first k entries of $\mathbf{T}_j$ contain $g - B_j$ 1's. Then the rectangle formed from the first k columns of $\mathbf{S}_j$ must be g-good, because its weight is at most $(g - B_j) + B_j$. The remaining part of $\mathbf{S}_j$ is also g-good, because its weight is at most $S_j - (g - B_j) < S_j - g/2 \leq g$. Again, we have $a_j = 2 < 2S_j/g$.

Case 3: $T_j > g$ and $2S_j/g > 3$. We can trivially partition $\mathbf{T}_j$ into $\ell > 1$ rectangles $\mathbf{T}_{j,1}, \mathbf{T}_{j,2}, \ldots, \mathbf{T}_{j,\ell}$ (going, say from left to right, columnwise), in which rectangles $\mathbf{T}_{j,1}, \mathbf{T}_{j,2}, \ldots, \mathbf{T}_{j,\ell-1}$ have weight exactly equal to g, and rectangle $\mathbf{T}_{j,\ell}$ has weight at most g. Then, our partition of $\mathbf{S}_j$ consists of the rectangles $\mathbf{B}_j$ and $\mathbf{T}_{j,1}, \mathbf{T}_{j,2}, \ldots, \mathbf{T}_{j,\ell}$. Obviously, all the rectangles are g-good. If this partition contains exactly 3 rectangles, then we clearly satisfy $a_j = 3 < 2S_j/g$. If it contains $a_j \geq 4$ rectangles, then we have $S_j > (a_j - 2)g \geq a_j g/2$, and thus $a_j < 2S_j/g$.

Note that after the initial computation of the slices we perform only the linear scans of the tops of the slices and this can be done easily in $O(n+m)$ time. ❑

Using Lemma 1, we can prove the following theorem[6].

Theorem 1 *The RTILE and DRTILE problems on a* $\{0,1\}$*-array can be approximated to within a factor of* 2 *in* $O(n+m)$ *time.*

[6] This theorem was also proved independently in a very recent conference paper [LP00, Theorems 2 and 3].

Proof. First, we prove the result for the RTILE problem. Assume that a given 0-1 array $\mathbf{A}$ can be partitioned into p rectangles, each of weight at most w. Obviously, $w \geq \lceil A/p \rceil$. We apply the algorithm of Lemma 1 for $g = \lceil 2A/p \rceil$ which yields a partition of $\mathbf{A}$ into at most $\lceil 2A/g \rceil \leq 2A/(2A/p) = p$ g-good rectangles. Because $g \leq 2\lceil A/p \rceil \leq 2w$, our approximation ratio equals 2.

Next, we prove the result for the DRTILE problem. Again, assume that a $\mathbf{A}$ can be partitioned into p rectangles, each of weight at most w. Obviously, $p \geq \lceil A/w \rceil$. We apply the algorithm of Lemma 1 for $g = w$. This yields a partition of $\mathbf{A}$ into $q \leq \lceil 2A/w \rceil$ g-good rectangles. Now, $p \geq \lceil A/w \rceil \geq \frac{1}{2}\lceil 2A/w \rceil \geq {}^{q}/_{2}$. ❑

3.3 The RTILE Problem on Arbitrary Arrays

Using the slicing technique in Section 3.1 with a novel method of combining slices and using more elaborate accounting of the number of rectangles used, we can prove the following theorem.

Theorem 2 *The RTILE problem on arbitrary arrays can be approximated to within a factor of* ${}^{11}/_{5}$ *in* $O(n+m)$ *time.*

Proof. Assume that the input array $\mathbf{A}$ can be partitioned into p rectangles of weight at most w. Our goal is to find a partition into p rectangles of weights at most $\frac{11}{5}w$.

Suppose that $x = W/p$ and y is the largest entry of $\mathbf{A}$. To avoid dealing with fractions in our proof, we rescale the entries of $\mathbf{A}$ and w, if necessary, such that $5 = \max(x, y)$. It is clear that after the rescaling we have $w \geq 5$. In the remaining part of the proof, we can therefore assume than no array entry exceeds 5. Moreover, to achieve an approximation ratio of ${}^{11}/_{5}$, it is sufficient to ensure that all rectangles in our collection are 11-good. Therefore it suffices to find a partition of $\mathbf{A}$ into at most $\lceil A/5 \rceil$ 11-good rectangles, or, equivalently, a partition of $\mathbf{A}$ into q 11-good rectangles where $5q - A < 5$.

We first partition $\mathbf{A}$ into slices (with $g = 11$) and compute all R_i's,S_j's, T_j's and B_j's in $O(n+m)$ time as outlined in Section 3.1. We next partition each slice separately. If $\mathbf{S}_j$ is partitioned into a_j 11-good rectangles, we say that $\mathbf{S}_j$ has a *deficit* of $d_j = 5a_j - S_j$. It suffices to partition the slices in such a way that $\sum_j d_j < 5$, because $\sum_j d_j = \sum_j (5a_j - S_j) = 5q - A$ (where $q = \sum_j a_j$ is the total number of rectangles). To improve readability and provide better insights into the ideas behind our approach, we provide the proofs of Lemmas 2, 3, 4 and 5 in the appendix.

Lemma 2 *Assume that* $T_j \leq 6b+5$ *for some positive integer* b. *Then we can partition* $\mathbf{T}_j$ *into* b 11*-good rectangles.*

Lemma 3 *Assume that* $T_j = 6b + 5 + y$ *for an integer* $b \geq 2$ *and a real* $y \leq 5$. *Then we can partition* $\mathbf{T}_j$ *either into* b 11*-good rectangles, or into* $b+1$ *rectangles each of weight at most* $6 + y$.

Lemma 4 *If* $S_j \geq 16$ *then we can partition the* j^{th} *slice* $\mathbf{S}_j$ *into* 11*-good rectangles with a deficit of* $d_j \leq -1$.

Lemma 5 *If* $T_j \leq 11$ *then we can partition* $\mathbf{S}_j$ *into* 11*-good rectangles with a deficit of* $d_j \leq -1$.

We now describe an algorithm that runs in time $O(n+m)$ and finds a partition of $\mathbf{A}$ into at most $\lceil A/5 \rceil$ rectangles (tiles) each of weight at most 11.

We first partition the input array into slices (with $g = 11$) as described in Section 3.1 in $O(n+m)$ time. The slices satisfying the assumption of Lemma 4 or Lemma 5 are partitioned into 11-good rectangles with the deficit of -1 or less in linear time. Below we describe how to handle the remaining slices (for which $T_j > 11$ and $S_j < 16$).

If $T_j > 11$ and $S_j < 16$, we make a preliminary partition of $\mathbf{S}_j$ into six rectangles. First, we partition $\mathbf{T}_j$ into three rectangles, from left to right $\mathbf{C}$, $\mathbf{D}$ and $\mathbf{E}$, so that the middle rectangle $\mathbf{D}$ consists of one entry only, while $C \le T_j/2$ and $E \le T_j/2$. Next, we partition $\mathbf{B}_j$ into $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{H}$ so that these rectangles align as shown in the side figure. Obviously, these rectangles can be found in linear time.

C	**D**	**E**
F	**G**	**H**

Before we proceed, we need to observe that rectangle $\mathbf{C} \cup \mathbf{F}$ is 11-good, and that by a symmetric argument, $\mathbf{E} \cup \mathbf{H}$ is 11-good as well. Indeed, $C + F \le S_j - (D+E) \le S_j - T_j/2 < 16 - 11/2 = 10\frac{1}{2}$.

Easy Case: $\mathbf{S}_j - (\mathbf{C} \cup \mathbf{F})$ is 11-good, or $\mathbf{S}_j - (\mathbf{E} \cup \mathbf{H})$ is 11-good. We can then partition $\mathbf{S}_j$ into two 11-good rectangles, $\mathbf{C} \cup \mathbf{F}$ and $\mathbf{S}_j - (\mathbf{C} \cup \mathbf{F})$, or $\mathbf{E} \cup \mathbf{H}$ and $\mathbf{S}_j - (\mathbf{E} \cup \mathbf{H})$, respectively. Because $S_j > 11$, we have a deficit of $d_j = 10 - S_j < -1$.

Hard Case: The easy case does not apply. We will show in a moment that $\mathbf{D} \cup \mathbf{G}$ is 11-good, so we can partition $\mathbf{S}_j$ into three 11-good rectangles $\mathbf{C} \cup \mathbf{F}$, $\mathbf{E} \cup \mathbf{H}$ and $\mathbf{D} \cup \mathbf{G}$. In this case the deficit $d_j = 15 - S_j$ can be positive, so we may be forced to change this initial partition. Therefore we need to characterize the weights of the six rectangles that are possible in the hard case.

Because the easy case does not apply, we can conclude that

$$\begin{aligned}
& 22 < (S_j - C - F) + (S_j - E - H) \\
& = (D + E + G + H) + (C + D + F + G) \\
& = S_j + D + G \\
& \le S_j + 5 + B_j \\
& < S_j + 5 + (S_j - 11) \\
\equiv\ & 28 < 2S_j \\
\equiv\ & S_j > 14
\end{aligned}$$

Therefore in the hard case we have $S_j = 14 + x$ for some real x, $0 < x < 2$, and the deficit is $d_j = 15 - S_j = 1 - x$. Hence, $-1 < d_j < 1$.

We can also show that in hard case the following inequalities are true:

(a) $B_j = F + G + H < 3 + x$;
(b) $C + F < 3 + x$ and $E + H < 3 + x$;
(c) $F < 2x$ and $H < 2x$.

Inequality **(a)** follows from $B_j = S_j - T_j = 14 + x - T_j < 14 + x - 11$.

Inequalities in **(b)** follow since $S_j - C - F > 11$ (because the easy case does not apply) and thus $C + F < S_j - 11 = 3 + x$ (and, by a symmetric argument $E + H < 3 + x$).

To prove the inequalities in **(c)**, suppose that one of them does not hold, w.l.o.g $F \geq 2x$; this and **(a)** implies $G < 3 - x$, thus $S_j - C - F = (C + F) + D + G < (3 + x) + 5 + (3 - x) = 11$, which means that $\mathbf{S}_j - C - F$ is good and the easy case applies, a contradiction.

By substituting $1 - d_j$ for x, we can rewrite **(a-c)** as follows:

(a') $B_j = F + G + H < 4 - d_j$;
(b') $C + F < 4 - d_j$ and $E + H < 4 - d_j$;
(c') $F < 2 - 2d_j$ and $H < 2 - 2d_j$.

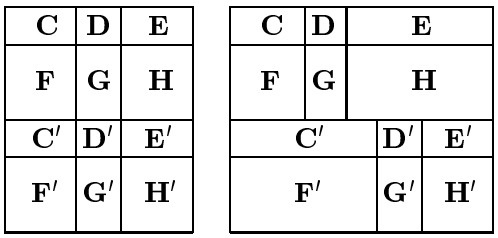
Let l be the number of slices produced by the slicing algorithm in Section 3.1, with $\mathbf{S}_l$ be the remainder slice (if present). Now for each $j = 1, \ldots, l-1$ we consider the accumulated deficit $\Delta_j = \sum_{k=1}^{j} d_k$. Our goal is to keep Δ_j small, even though some terms in this sum, those that correspond to the hard cases, may be positive. Consider the smallest j such that $\Delta_j \geq 1$. Because each deficit contributing to Δ_j is less than 1, this implies that $d_j > 0$ and $d_j + d_{j-1} > 0$.

Our strategy is that whenever $\Delta_j \geq 1$, we partition $\mathbf{S}_{j-1} \cup \mathbf{S}_j$ again, this time into 5 rectangles rather than 6. This repartition subtracts 5 from the accumulated deficit, so it drops below -3. This way we never let Δ_j stay above 1 for $j < l$. To finish, we will need to account separately for the remainder slice $\mathbf{S}_l$ that can have a deficit larger than 1.

To account for the remainder slice $\mathbf{S}_l$, consider first the case when the slice $\mathbf{S}_{l-1}$ is *not* an example of the hard case. Then $\Delta_{l-1} = \Delta_{l-2} + d_{l-1} < 1 - 1 = 0$, hence $\Delta_l < d_l < 5$. On the other hand, if $\mathbf{S}_{l-1}$ is an example of the hard case, then we can partition it into $\mathbf{C} \cup \mathbf{F}$ (a rectangle of weight at most $4 - d_{l-1} < 5$), $\mathbf{E} \cup \mathbf{H}$ (again a rectangle of weight at most $4 - d_{l-1} < 5$), and $\mathbf{D} \cup \mathbf{G}$ (a rectangle of weight at most $5 + (3 - x) = 5 + 4 - d_{l-1} < 10$). If $S_l \leq 1$, then we can extend these three rectangles vertically up to cover $\mathbf{S}_l$; if $S_l > 1$, then $d_l = 5 - S_l < 4$ and $\Delta_l < 1 + 4 = 5$, hence we are allowed to cover S_l itself by one rectangle. Notice that all cases are easily implementable in linear time.

To finish the proof, we return to the case when $\Delta_j > 1$ and hence we need to partition $\mathbf{S}_{j-1} \cup \mathbf{S}_j$ again. Because $0 < d_j < 1$ and $d_{j-1} > -d_j$, both the slices $\mathbf{S}_{j-1}$ and $\mathbf{S}_j$ are examples of the hard case. We partition $\mathbf{S}_j$ into $\mathbf{C}$, $\mathbf{D}$, $\mathbf{E}$, $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{H}$and $\mathbf{S}_{j-1}$ into $\mathbf{C}'$, $\mathbf{D}'$, $\mathbf{E}'$, $\mathbf{F}'$, $\mathbf{G}'$ and $\mathbf{H}'$, in the manner shown in beginning of the proof.

Because $\mathbf{D}$ and $\mathbf{D}'$ are single elements (and, consequently $\mathbf{G}$ and $\mathbf{G}'$ are each a single column), we must have one of the configurations shown in the side figure, or a symmetric variation of them. In the left configuration, the right vertical boundary of $\mathbf{C}'$ aligns with the right vertical boundary of $\mathbf{C}$, and in the right configuration, the right vertical boundary of $\mathbf{C}'$ aligns or to the right of the right vertical boundary of $\mathbf{G}$.



In the left configuration, we can use four rectangles, the left columns of four arrays forms one rectangle $\mathbf{C} \cup \mathbf{F} \cup \mathbf{C}' \cup \mathbf{F}'$, the right columns forming another rectangle $\mathbf{E} \cup \mathbf{H} \cup \mathbf{E}' \cup \mathbf{H}'$, while the remaining rectangle are $\mathbf{D} \cup \mathbf{G}$ and $\mathbf{D}' \cup \mathbf{G}'$. The weights of these arrays can be estimated as follows:

$C + F + C' + F' < 4 - d_j + 4 - d_{j-1} = 8 - (d_j + d_{j-1}) < 8$;
$E + H + E' + H' < 8$ by a symmetric argument;
$D + G < 5 + B_j = 5 + 4 - d_j < 9$;

$D' + G' < 5 + B_{j-1} = 5 + 4 - d_{j-1} < 10$.

In the right configuration we can use five arrays. The first is $\mathbf{B}_{j-1}$. The second is $\mathbf{C} \cup \mathbf{D}$ of weight at most $4 - d_j + 5 = 9 + d_j < 9$. The third is $\mathbf{E}$. The fourth is $\mathbf{C}'$ extended upwards, to contain $\mathbf{F}$, $\mathbf{G}$ and a fragment of $\mathbf{H}$; its weight is at most $4 - d_{j-1} + B_j \leq 8 - (d_{j-1} + d_j) < 8$. The last one is $\mathbf{D}' \cup \mathbf{E}'$ extended upwards to contain the remaining fragment of $\mathbf{H}$; its weight is at most $5 + 4 - d_{j-1} + 2 - 2d_j = 11 - (d_{j-1} + d_j) - d_j < 11$. In both cases, it is easy to compute the new partition in linear time. This completes the proof of Theorem 2. ❑

3.4 The DRTILE Problem on Arbitrary Arrays in d Dimensions

In this section, we turn our attention to the DRTILE problem in d-dimensions for $d \geq 1$. Let $\mathbf{A} = [a_{i_1,i_2,\ldots,i_d}]$ be the d-dimensional input array where $1 \leq i_1, i_2, \ldots, i_d \leq n$. An obvious lower bound on the number of tiles needed is $\lceil A/w \rceil$. In order to have better approximation ratios, we use another lower bound for this problem, extending the concept of *anti-rectangle sets* for the interior cover problem for rectilinear polygons [BD97].

Definition 2 *Two elements $a_{i_1,i_2,\ldots,i_d}$ and $a_{j_1,j_2,\ldots,j_d}$ of $\mathbf{A}$ form anti-rectangle pair if they cannot be covered together by one hyper-rectangle with weight at most w. An anti-rectangle set S for the array $\mathbf{A}$ is a set of elements of $\mathbf{A}$ such that every two elements of S form an anti-rectangle pair.*

Since no two elements of an anti-rectangle set for $\mathbf{A}$ can be covered together by a hyper-rectangle of weight at most w, the following observation is obvious and provides our second lower bound.

Observation 1 *If S is an anti-rectangle set for $\mathbf{A}$, then any solution to the DRTILE problem for $\mathbf{A}$ needs at least $|S|$ hyper-rectangles.*

We use the following two definitions later in the proofs.

Definition 3 *An array is x-bounded (for some $x \geq 0$) if every element of the array is at most x.*

Definition 4 *Assume that $\mathbf{B}$ is a d-dimensional array where the the d^{th} index ranges between 1 and n_d. The projection of $\mathbf{B} = [b_{i_1,\ldots,i_{d-1},i_d}]$ is the $(d-1)$-dimensional array $\overline{\mathbf{B}} = [\bar{b}_{i_1,\ldots,i_{d-1}}]$ defined by the equation:*

$$\bar{b}_{i_1,\ldots,i_{d-1}} = \sum_{i_d=1}^{n_d} b_{i_1,\ldots,i_{d-1},i_d}$$

Using a slicing mechanism somewhat different than the one described in Section 3.1 and a combination of two above lower bounds, we can prove the following theorem.

Theorem 3 *The DRTILE problem on arbitrary arrays in d dimensions ($d \geq 1$) can be approximated to within a factor of $2d - 1$ in $O(d(m+n))$ time.*

Proof. For simplicity of notation, we will assume, without loss of generality, that $w = 1$ (if the weight limit is different, we simply rescale the entries of $\mathbf{A}$). This implies that $\mathbf{A}$ is 1-bounded. Let k^* be the minimum number of rectangles in a partition of the input array $\mathbf{A}$ of total weight A such that each rectangle has weight no more than 1. Obviously, $k^* \geq A$.

For $d = 1$, we solve the DRTILE problem exactly using a simple greedy method in $O(n)$ time [KMP98]. For $d > 1$, our algorithm uses the algorithm for dimension $d-1$ in the following way[7]:

(a) We find the largest index $k_1 \leq n$ such that the array $\mathbf{A}_1$ formed from the entries $[a_{i_1}, a_{i_2}, \ldots, a_{i_d}]$ of $\mathbf{A}$ with $i_d \in (k_0 = 0, k_1]$ satisfies the condition that $\overline{\mathbf{A}_1}$ is 1-bounded; if $k_1 < n_d$, we find the largest $k_2 \leq n$ such that the array $\mathbf{A}_2$ formed from entries of $[a_{i_1}, a_{i_2}, \ldots, a_{i_d}]$ of $\mathbf{A}$ with $i_d \in (k_1, k_2]$ satisfies the condition that $\overline{\mathbf{A}_2}$ is 1-bounded, and so on. In this way we partition $\mathbf{A}$ into s *slices* $\mathbf{A}_1, \ldots, \mathbf{A}_s$.

(b) For $1 \leq r \leq s$ apply the $(d-1)$-dimensional algorithm to $\overline{\mathbf{A}}_r$, the projection of the r^{th} slice (array) $\mathbf{A}_r$. As a result, we will obtain, for each r, a set of hyper-rectangles forming a partition of $\overline{\mathbf{A}}_r$.

(c) For $1 \leq r \leq s$ replace each hyper-rectangle $\overline{\mathbf{B}}$ from the partition of $\overline{\mathbf{A}}_r$ with the hyper-rectangle $\mathbf{B} = \overline{\mathbf{B}} \times [k_{r-1} + 1, k_r]$. All these hyper-rectangles together form our solution to the DRTILE problem for $\mathbf{A}$.

Let t be the total number of hyper-rectangles produced by the above algorithm. First we show that our algorithm produces a correct solution.

Lemma 6 *Every hyper-rectangle produced by our algorithm has a total weight of at most* 1.

Proof. We prove by induction on d. For $d = 1$ the claim is obvious. Consider the case when $d > 1$. The total weight of the partition $\overline{\mathbf{B}}$ in $\overline{\mathbf{A}}_r$ is at most 1 by inductive hypothesis, since $\overline{\mathbf{A}}_r$ is a $(d-1)$-dimensional array. Hence,

$$\sum_{b_{i_1,\ldots,i_{d-1},i_d} \in \mathbf{B}} b_{i_1,\ldots,i_{d-1},i_d} = \sum_{\bar{b}_{i_1,\ldots,i_{d-1}} \in \overline{\mathbf{B}}} \bar{b}_{i_1,\ldots,i_{d-1}} \leq 1$$

❑

Next we turn our attention to prove the promised approximation bound of our algorithm.

Lemma 7 $k^* \geq s$

Proof. We will show that there is an anti-rectangle set of size at least s for $\mathbf{A}$. Notice that, for $1 \leq r < s$, k_r is the largest index in $(k_{r-1}, n]$ such that $\overline{\mathbf{A}_r}$ is 1-bounded. Consequently, for each $1 \leq r < s$, we can find an index vector $\iota_r = (\iota_{r,1}, \iota_{r,2}, \ldots, \iota_{r,d-1})$ such that the entry $\bar{a}_{\iota_r}$ of $\overline{\mathbf{A}}_r$ would exceed 1 if we would extend the range of the last index in $\overline{\mathbf{A}}_r$ from $(k_{r-1}, k_r]$ to $(k_{r-1}, k_r + 1]$. Define ι_s as an index vector of an arbitrary entry of A_s. Now, for $1 \leq r \leq s$, we define a_r to be the entry of $\mathbf{A}$ with index vector $r = (\iota_r, k_{r-1} + 1)$. The set $\{b_r \mid 1 \leq r \leq s\}$ of entries of $\mathbf{A}$ form an anti-rectangle set for $\mathbf{A}$. ❑

[7] The notation $(x, y]$ denotes the set of integers $\{z \mid x < z \leq y\}$ for two integers x and y

Lemma 8 *The weight A of $\mathbf{A}$ satisfies $2dA \geq t - 1$.*

Proof. By induction on d. First, consider the case when $d = 1$. Then, $t = k^*$. Since the sum of weights of every two consecutive intervals in any optimal solution must be larger than 1, we have $k^* \leq 2A + 1$.

Next, we prove the inductive step. Assume that the r^{th} slice A_r was partitioned into t_r tiles. By applying the inductive hypothesis to the projections of the slices of $\mathbf{A}$ we obtain

$$2(d-1)A = 2(d-1)\sum_{r=1}^{s} \overline{A}_r \geq \sum_{r-1}^{s}(t_r - 1) = t - s$$

On the other hand, the sum of weights of every consecutive pair of slices exceeds 1; otherwise the projection of their union would be 1-bounded, a contradiction. Thus

$$2A > s - 1$$

We obtain the claim by adding these two inequalities. ❑

Lemma 9 $t \leq 2dk^*$

Proof. By applying Lemma 8 to the to the projections of the slices of $\mathbf{A}$ we obtained the inequality $t - s \leq 2(d-1)A \leq 2(d-1)k^*$. Lemma 7 yields the inequality $s \leq k^*$. The claim follows form adding these two inequalities. ❑

Finally we show the time-complexity of our algorithm. Since after Step **(a)** the projections of different slices are disjoint, it suffices to show how to find the indices $k_1, k_2, \ldots, k_s$ and the projections of all the slices in $O(m+n)$ time, assuming that $\mathbf{A}$ is given in its standard sparse representation as described in Section 2. For each index j in the d^{th} dimension of $\mathbf{A}$ $(1 \leq j \leq n)$, let L_j be the list of $(d+1)$-tuples $(i_1, i_2, \ldots, i_{d-1}, j\ ;\ \mathbf{A}[i_1, i_2, \ldots, i_{d-1}, j])$ in any arbitrary order. The lists $L_1, L_2, \ldots, L_n$ can be computed in a total of $O(m+n)$ time simply by traversing each of the m $(d+1)$-tuples of $\mathbf{A}$ and inserting these tuples in the appropriate L_j based on their d^{th} coordinate. Since the $(d+1)$-tuples of $\mathbf{A}$ are given in a lexicographically sorted order on their first d coordinates, it is easy to find in $O(m)$ time the at most m $(d-1)$-tuples $(i_1, i_2, \ldots, i_{d-1})$, where for every such tuple there is some j and x such that $(i_1, i_2, \ldots, i_{d-1}, j\ ;\ x)$ is a $(d+1)$-tuple of $\mathbf{A}$. Next, we maintain an m-element array $\mathbf{B}$ corresponding to these at most m $(d-1)$-tuples, where a tuple $(i_1, i_2, \ldots, i_{d-1})$ correspond to the array element $\mathbf{B}[i_1, i_2, \ldots, i_{d-1}]$. Every element of $\mathbf{B}$ also has an auxiliary field, which is either *uninitialized* or *initialized by a positive integer*. Initially, the auxiliary field of $\mathbf{B}[i_1, i_2, \ldots, i_{d-1}]$ is set to uninitialized for every tuple $(i_1, i_2, \ldots, i_{d-1})$, and obviously this takes $O(m)$ time. We also set a variable i to 1 (this will indicate that we are currently trying to find the value of k_i), and a variable *bound* to -1. Now, we do the following to find the index k_1:

- We examine the lists $L_1, L_2, L_3, \ldots$ in this order and for each entry $(i_1, i_2, \ldots, i_{d-1}, j\ ;\ x)$ in L_j,
  - if the auxiliary field of $\mathbf{B}[i_1, i_2, \ldots, i_{d-1}]$ is uninitialized or less than i, then we set $\mathbf{B}[i_1, i_2, \ldots, i_{d-1}]$ to x, set the auxiliary field of $\mathbf{B}[i_1, i_2, \ldots, i_{d-1}]$ to i, and set bound to be $\max\{\ bound, x\ \}$.

– otherwise, we increment $\mathbf{B}[i_1, i_2, \ldots, i_{d-1}]$ by x and set bound to be $\max\{\ bound, \mathbf{B}[i_1, i_2, \ldots, i_{d-1}]\ \}$.

- We stop as soon as we arrive at an index j such that, while the corresponding list L_j is being processed, *bound* exceeds 1. Then, $k_1 = j - 1$.

We increase i by 1, reset *bound* to -1, and continue the same procedure starting from the first entry in list L_j to find k_2, and so on. This takes a total of $O(n+m)$ time. Using a similar approach, we can also compute the projections of each slice A_r in $O(n+m)$ time. This completes the proof of Theorem 3. ❑

4 Approximating d-RPACK

In this section we first give a simple approximation algorithm for d-RPACK with approximation ratio $\lceil \log(n+1) \rceil^{d-1}$, and then show how to further improve the performance ratio of the algorithm at the expense of increasing the running time. We assume, without loss of generality, that $p = \Omega(n)$.

4.1 $(\lfloor 1 + \log n \rfloor)^{d-1}$-approximation algorithm for d-RPACK

The main result of this section is the following theorem.

Theorem 4 *The d-RPACK problem can be approximated to within a factor of $(\lfloor 1 + \log n \rfloor)^{d-1}$ of the optimum in $O(dp \log^{\varepsilon} p + dn \frac{\log n}{\log \log n} + pk)$ time (for any constant $\varepsilon > 0$).*

In the remaining part of this section, we prove Theorem 4. We will use the following notations in the description of our algorithm and proof. For any d-dimensional hyper-rectangle $D = \times_{i=1}^{d} [a_i, b_i]$, let $D + \alpha$ denote the d-dimensional hyper-rectangle $\times_{j=1}^{d} [a_i + \alpha, b_i + \alpha]$, and if $d > 1$, let $\overline{D}$, termed as the *projection* of D, denote the $(d-1)$-dimensional hyper-rectangle $\times_{j=1}^{d-1} [a_i, b_i]$. We extend these notations to an arbitrary set X of d-dimensional hyper-rectangles by defining $X + \alpha = \{x + \alpha \mid x \in X\}$ and $\overline{X} = \{\overline{x} \mid x \in X\}$. For a set of hyper-rectangles X, let $|X|$ denote the number of hyper-rectangles in X.

Let $D_1, D_2, \ldots, D_p$ be the set of p weighted d-dimensional input rectangles, where $D_i = \times_{j=1}^{d} [b_{i,j}, e_{i,j}]$ with $b_{i,j}, e_{i,j} \in \{1, 2, \ldots, n\}$ for all i and j. We note that the 1-RPACK problem, which we call as the *Interval Packing* (IPACK) problem, is easily solvable in $O(pk)$ time via dynamic programming technique [KMP98][8]. For notational convenience, let $q = \lfloor 1 + \log n \rfloor$. We will prove the result by induction on d. For $d = 1$, the result follows directly via solution of IPACK. Inductively, assume that we have an algorithm for $(d-1)$-RPACK with the time and approximation bounds as stated in the theorem, and consider the d-RAPCK problem on the set of hyper-rectangles R for $d > 1$. Our approach will be as follows:

- We will divide R, in $O(p \log^{\varepsilon} p + n \frac{\log n}{\log \log n})$ time, into at most q *disjoint* collections of hyper-rectangles $R_1, R_2, \ldots, R_q$ such that an optimum solution of the d-RPACK problem on each set R_i is also an optimum solution of the $(d-1)$-RPACK problem on $\overline{R_i}$.

[8] Since the endpoints of the p intervals are in $\{1, 2, \ldots, n\}$, they can be sorted in $O(p + n) = O(p)$ time

- By inductive hypothesis, we can approximate the $(d-1)$-RPACK problem on each $\overline{R_i}$ separately within a factor of $(\lfloor 1+\log n \rfloor)^{d-1}$ of the optimum for each $\overline{R_i}$.
- By pigeonhole principle, the best of all these solutions is within a factor of $q(\lfloor 1+\log n_i \rfloor)^{d-1} = (\lfloor 1+\log n_i \rfloor)^{d}$ of the optimum solution.
- The set of various 1-RPACK problems at the base level of recursion (corresponding to $d=1$) can be solved in a total of $O(pk)$ time since there are in all p intervals none of which appear in more than one set and the endpoints of all these p intervals divided into at most $\min\{p, (\lfloor 1+\log n_i \rfloor)^{d-1}\}$ groups can be sorted in a total of $O(p+n) = O(p)$ time. Hence, the total time taken our algorithm will be

$$\begin{aligned} & O\left(\textstyle\sum_{i=1}^{q} |R_i|(d-1)\log^{\varepsilon}|R_i| + (d-1)n\frac{\log n}{\log\log n} + \sum_{i=1}^{k}|R_i|k + p\log^{\varepsilon} p + n\frac{\log n}{\log\log n}\right) \\ = \; & O\left(p(d-1)\log^{\varepsilon} p + (d-1)n\frac{\log n}{\log\log n} + pk + p\log^{\varepsilon} p + n\frac{\log n}{\log\log n}\right) \\ = \; & O\left(dp\log^{\varepsilon} p + dn\frac{\log n}{\log\log n} + pk\right) \end{aligned}$$

Now, we show how to construct the sets $R_1, R_2, \ldots, R_q$. First, we need the following definitions and results.

Definition 5 (a-overlap) *A set X of d-dimensional hyper-rectangles is called a-overlapping if and only if there is an integer $a \in \{1,2,\ldots,n\}$ such that for any rectangle $X_i = \times_{\ell=1}^{d}[b_{i,\ell}, e_{i,\ell}]$ in X, we have $b_{i,d} \leq a \leq e_{i,d}$.*

A set of a-overlapping hyper-rectangles allows us to reduce the dimension of the d-RPACK problem by 1, as shown by the two lemmas below.

Lemma 10 *Let X be any collection of d-dimensional a-overlapping hyper-rectangles. Then, any two rectangles $X_i, X_j \in X$ intersect if and only if their projections $\overline{X_i}$ and $\overline{X_j}$ intersect.*

Proof. Obviously, if X_i and X_j intersect, then $\overline{X_i}$ and $\overline{X_j}$ intersect also. Conversely, suppose that $\overline{X_i}$ and $\overline{X_j}$ intersect and let $\mathbf{x} \in [1,n]^{d-1}$ be a $(d-1)$-dimensional point in their intersection. Then, the d-dimensional point $(\mathbf{x}, a)$ belongs to both X_i and X_j, thus X_i and X_j intersect also. ❑

Lemma 11 *Let X be any collection of d-dimensional a-overlapping hyper-rectangles. Then, an optimal solution of the d-RPACK problem for X is also an optimal solution of the $(d-1)$-RPACK problem for $\overline{X}$ and vice versa.*

Proof. By Lemma 10, the solution to the d-RPACK problem on X is identical to the solution of the $(d-1)$-RPACK problem on $\overline{X}$. ❑

Corollary 1 *An instance X of the 2-RPACK problem in which rectangles in X are a-overlapping can be solved exactly in $O(pk)$ time.*

Proof. By Lemma 10, an optimal solution to the 2-RPACK problem on X is identical to a corresponding optimal solution of the IPACK problem on the intervals in $\overline{X}$, which can be solved in $O(pk)$ time [KMP98]. ❑

Definition 6 ($(\beta_1, \beta_2, \ldots, \beta_{m-1})$-separation) *Let $X_1, X_2, \ldots, X_m$ be m disjoint groups of d-dimensional hyper-rectangles. Then, $X_1, X_2, \ldots, X_m$ are $(\beta_1, \beta_2, \ldots, \beta_{m-1})$-separated if and only if there exists real numbers $\beta_1, \beta_2, \ldots, \beta_{m-1}$ such that for any m hyper-rectangles $r_1, r_2, \ldots, r_m$, with $r_i = \times_{j=1}^{d}[b_{i,j}, e_{i,j}] \in X_i$ for $1 \leq i \leq m$, the following inequalities hold:*

$$
\begin{aligned}
& e_{1,d} < \beta_1 \\
& \beta_{m-1} < b_{m,d} \\
& \beta_j < b_{j+1,d} \leq e_{j+1,d} < \beta_{j+1} \quad \textit{for all } 1 \leq j \leq m-2
\end{aligned}
$$

The following observation is easy.

Observation 2 *Let $X_1, X_2, \ldots, X_m$ be m disjoint groups of d-dimensional hyper-rectangles. For any two hyper-rectangles $x \in X_i$ and $y \in X_j$, x and y do not intersect if $i \neq j$.*

The definition of $(\beta_1, \beta_2, \ldots, \beta_{m-1})$-separation allows us to compute the solution of the d-RPACK problem for each group X_i by appropriately "translating" their d^{th} coordinate, as stated more precisely in the lemma below.

Lemma 12 *Let $X_1, X_2, \ldots, X_m$ be m disjoint groups of d-dimensional hyper-rectangles satisfying the following properties:*

(a) *$X_1, X_2, \ldots, X_m$ are $(\beta_1, \beta_2, \ldots, \beta_{m-1})$-separated.*

(b) *hyper-rectangles in each group X_i are a_i-overlapping for some a_i.*

(c) *the endpoints of the d^{th} dimension of all the hyper-rectangles in all groups are from the set $\{1, 2, \ldots, n\}$.*

Then, an optimum solution of the d-RPACK problem for $\cup_{i=1}^{m} X_i$ is also an optimum solution of the $(d-1)$-RPACK problem for $\cup_{i=1}^{m} \overline{X_i + i(n+1)}$ and vice versa.

Proof. It is sufficient to show that the intersection properties of the hyper-rectangles do not change by the above transformation. Let $x = \times_{\ell=1}^{d}[b_{i,\ell}, e_{i,\ell}] \in X_i$ and $y = \times_{\ell=1}^{d}[b_{j,\ell}, e_{j,\ell}] \in X_j$ be two hyper-rectangles from two distinct groups X_i and X_j. Assume, without loss of generality, that $i < j$. Then, by Observation 2, x and y do not intersect. Because $X_1, X_2, \ldots, X_m$ are $(\beta_1, \beta_2, \ldots, \beta_{m-1})$-separated, $e_{i,d} < \beta_i < b_{j,d}$. This implies that $\overline{x + i(n+1)}$ and $\overline{y + j(n+1)}$ do not intersect either, since in particular,

$$b_{j,d} + j(n+1) - e_{i,d} + i(n+1) = (b_{j,d} - e_{i,d}) + (n+1)(j-i) > 0$$

where the last inequality follows from $(b_{j,d} - e_{i,d}) < n$.

On the other hand, consider two hyper-rectangles $x, y \in X_i$ for some i. Since x and y are a-overlapping, $\overline{x + i(n+1)}$ and $\overline{y + i(n+1)}$ are $a + i(n+1)$-overlapping. Hence, by Lemma 10, x and y

intersect if and only if $\overline{x + i(n+1)}$ and $\overline{y + i(n+1)}$ intersect. As a result, $\cup_{i=1}^{m} \overline{X_i + i(n+1)}$ preserves the intersection property of $\cup_{i=1}^{m} X_i$ and the result follows. ❑

For future usage, for a set X of d-dimensional hyper-rectangles and a real number a, let us define the following disjoint subsets of X:

$$\begin{aligned} INT(a, X) &= \{r \mid r = \times_{j=1}^{d} [b_j, e_j] \in X \textbf{ and } b_d \leq a \leq e_d\} \\ ABOVE(a, X) &= \{r \mid r = \times_{j=1}^{d} [b_j, e_j] \in X \textbf{ and } a < b_d\} \\ BELOW(a, X) &= \{r \mid r = \times_{j=1}^{d} [b_j, e_j] \in X \textbf{ and } e_d < a\} \end{aligned}$$

The following observations are obvious:

($\star$) Hyper-rectangles in $Int(a, X)$ are a-overlapping.

($\star\star$) For any two subsets $S_1 \subseteq Above(a, X)$ and $S_2 \subseteq Below(a, X)$, S_1 and S_2 are a-separated.

Let T be a rooted *complete* binary tree of height $q - 1$ (the root is at height 0). Let $\ell(x), r(x)$ and $p(x)$ denote the left child, the right child and the parent of a node x in T, respectively, if they are present. To facilitate further discussion, we also identify each node of T with a string in $\{0, 1\}^*$ recursively starting from the root in the following manner:

- the root is the empty string ϵ,
- for a node $x \in \{0, 1\}^*$ of T, $\ell(x)$ and $r(x)$ are the strings $x0$ and $x1$, respectively.

Note that, for any node $x \in \{0, 1\}^*$ in T, $0 \leq |x| < q$. For two strings $x, y \in \{0, 1\}^*$, we will write $x \prec y$ to denote the fact that value of binary string x is less than the value of the binary string y, and let $|x|$ denote the length (i.e., number of zeroes and ones) of x. Each node x of T has also the following attributes:

- $S_x, C_x \subseteq R$, two subsets of the given set of hyper-rectangles R,
- $I_x = [a_x, b_x] \subseteq [1, n]$, an interval, and
- $H_x = \lceil \frac{a_x + b_x}{2} \rceil$, a real number between 1 and n.

The above attributes are computed, going level-by-level in T starting at the root, for each node x in the following manner:

- for the root, $S_\epsilon = INT(\lceil \frac{1+n}{2} \rceil, R)$, $C_\epsilon = R - S_\epsilon$, $I_\epsilon = [1, n]$, and $H_\epsilon = \lceil \frac{n+1}{2} \rceil$,
- let x be a node for which S_x, C_x, $I_x = [a_x, b_x]$ and H_x has already been computed. Then, if $S_x = \emptyset$, then S_{x0} and S_{x1} are both set to be $\emptyset$. Otherwise, set (for the left child of x) $I_{x0} = [a_x, H_x - 1]$, $H_{x0} = \lceil \frac{a_x + H_x - 1}{2} \rceil$, $S_{x0} = INT(BELOW(S_x, H_x), H_{x0})$, $C_{x0} = BELOW(S_x, H_{x0}) - S_x$, and (for the right child of x) $I_{x1} = [H_x + 1, b_x]$, $H_{x1} = \lceil \frac{H_x + 1 + b_x}{2} \rceil$, $S_{x1} = INT(ABOVE(S_x, H_x), H_{x1})$, $C_{x0} = BELOW(S_x, H_{x1}) - S_x$,

Our sets $R_1, R_2, \ldots, R_q$ are now defined by $R_i = \cup_{x \in \{0,1\}^*, |x|=i-1} S_x$. In other words, R_i consists of the union of all the S_x's at the i^{th} level of T. The following assertions hold:

(a) $\cup_x H_x = \{1, 2, \ldots, n\}$. Consider any $i \in \{1, 2, \ldots, n\}$. We start at the root of T. If $i = \lceil \frac{1+n}{2} \rceil$, then H_ϵ is i. Otherwise, if $i < \lceil \frac{1+n}{2} \rceil$, we recursively search for i in the left subtree of the root, else we recursively search for i in the right subtree of the root. Since the range for I_{x0} and I_{x1} are less than half the range of I_x for any node x and the height of T is $q-1$, we can find i to be equal to some H_x.

(b) For any two nodes x and y of T, $S_x \cap S_y = \emptyset$. As a results, the sets $R_1, R_2, \ldots, R_q$ are also mutually disjoint.

(c) By (a) above, and the definition of S_x for any node x, $\cup_{x \in T} S_x = R$. As a result, $\cup_{i=1}^{q} R_i = R$.

(d) For any node x, hyper-rectangles in S_x are H_x-overlapping (by definition of S_x and observation $(\star)$ above). By Lemma 11, a direct consequence of this is that an optimal solution of the d-RPACK problem for R_1 is also an optimal solution of the $(d-1)$-RPACK problem for $\overline{R_1}$ and vice versa.

(e) For $i > 1$, let $S_{x_1}, S_{x_2}, \ldots, S_{x_{2^{i-1}}}$ be the 2^{i-1} sets of hyper-rectangles whose union is R_i, where $x_j \in \{0,1\}^*$ and $|x_j| = i-1$ for all j, and $x_1 \prec x_2 \prec x_3 \prec \ldots \prec x_{2^{i-1}}$. Then, the sets $S_{x_1}, S_{x_2}, \ldots, S_{x_{2^{i-1}}}$ are $(\beta_1, \beta_2, \ldots, \beta_{m-1})$-separated for some $\beta_1, \beta_2, \ldots, \beta_{m-1}$.

Let x_j and x_{j+1} be any two consecutive strings (in the above order) over $\{0,1\}^*$ of length exactly $i-1$. Considering the first place from left where the two strings differ, let $x_j = y0z$ and $x_{j+1} = y1z'$, where $|z| = |z'|$. Then, the sets S_{x_j} and $S_{x_{j+1}}$ are H_y-separated. Finally, note that if $x_j \prec x_{j+1} \prec x_{j+2}$ are three consecutive strings over $\{0,1\}^*$ of length exactly $i-1$, and x_j and x_{j+1} (respectively, x_{j+1} and x_{j+2}) are H_{y_1}-separated (respectively, H_{y_2}-separated), then (i) either $|y_1| < |y_2|$, or (ii) $|y_1| = |y_2|$ and $y_1 \prec y_2$.

By **(d)** above, an optimal solution of the d-RPACK problem for R_1 is also an optimal solution of the $(d-1)$-RPACK problem for $\overline{R_1}$ and vice versa. By **(d)** and **(e)** above, for $i > 1$, the set R_i us the union of the sets $S_{x_1}, S_{x_2}, \ldots, S_{x_{2^{i-1}}}$, where $S_{x_1}, S_{x_2}, \ldots, S_{x_{2^{i-1}}}$ are $(\beta_1, \beta_2, \ldots, \beta_{m-1})$-separated and each S_{x_j} is H_j-overlapping. Hence, by Lemma 12, an optimum solution of the d-RPACK problem for R_i is also an optimum solution of the $(d-1)$-RPACK problem for $\overline{R_i}$ and vice versa.

It only remains to show how to find the sets $R_1, R_2, \ldots, R_q$ in $O(p \log^{\varepsilon} p + n \frac{\log n}{\log \log n})$ time. Note that there is no need to maintain explicitly the C_x sets for all nodes, since each R_i can be fully recovered from the corresponding S_x sets. It is trivial to compute the I_x and H_x sets for all nodes x in a total of $O(p)$ time. To compute the S_x sets for various nodes x, we first solve the following problem: given a set $\mathcal{I}$ of at most p intervals with endpoints of intervals in $\{1, 2, \ldots, n\}$, design a data structure $\mathcal{D}$ which will support the following operation (*stabbing query with deletion*):

> given a query number x, find (report) the set of intervals $\mathcal{I}_1 = \{[y, z] \in \mathcal{D} \mid y \leq x \leq z\}$, and delete these intervals in $\mathcal{I}_1$ from $\mathcal{D}$.

We use the *interval trie* data structure for $\mathcal{D}$ as described in [O88, Section 6]. Interval trie supports a stabbing query (without deletion) in time $O(k + \frac{\log n}{\log \log n})$ where k is the number of reported answers,

and supports update (insertion or deletion a single interval) in time $O(\log^{\varepsilon} p)$, where $\varepsilon > 0$ is a constant. Now, we perform the following:

- Initialize $\mathcal{D}$ to be empty. Then, for each $r = \times_{i=1}^{d}[a_i, b_i] \in R$, insert the interval $[a_d, b_d]$ in $\mathcal{D}$. This takes a total of $O(p \log^{\varepsilon} p)$ time.
- Let L be an ordering of the nodes in T in which we start at the root, visit T level by level and in each level visit the nodes in left to right order. It is easy to compute the ordering L in $O(p)$ time. Now, we examine the nodes in L in left to right order, and for every node x, we perform a stabbing query with deletion in $\mathcal{D}$ with query number H_x. The answer to this query is the set S_x. let k_x be the number of reported answers for x. Then, the total time taken for all the stabbing queries with deletion is $O(\sum_x k_x + n\frac{\log n}{\log\log n} + \sum_x k_x \log^{\varepsilon} p) = O(p \log^{\varepsilon} p + n\frac{\log n}{\log\log n})$.

This completes the proof of Theorem 4.

4.2 Further Improved Approximation Ratio for d-RPACK

We assume, without loss of generality, that $p \geq k > 1$ since if $k = 1$ then the problem can be trivially solved in $O(p)$ time.

Theorem 5 *For every $L \geq 2$, the d-RPACK problem can be approximated to within a factor of $(\lfloor 1 + \log_L n \rfloor)^{d-1}$ of the optimum in $O(p^{(L-1)^{d-1}+1}\, dk)$ time.*

Setting $L = 2^c$ in the above theorem, we obtain:

Corollary 2 *For every $c \geq 1$, there is an approximation algorithm for d-RPACK that runs in $O(p^{(2^c-1)^{d-1}+1}\, dk)$ time with an approximation ratio of $\left(\left\lfloor 1 + \frac{\log n}{c} \right\rfloor\right)^{d-1}$.*

Proof of Theorem 5. In the following discussions, each hyper-rectangle r has the following attributes: weight $w(r)$ and, for coordinate $i \in [1, d]$, starting and terminating values $s_i(r)$ and $t_i(r)$ (i.e., $r = \times_{i=1}^{d}[s_i(r), t_i(r)]$).

To formulate our approximation algorithm, we will first define a very special subproblem of d-RPACK which has a polynomial time solution. Analogous to the definition of a-overlap of hyper-rectangles (Definition 5 in the previous section) we define an instance of the d-RPACK problem to be *A-restricted* for some $A = \times_{i=2}^{d}\{a_{i_1}, a_{i_2}, \ldots, a_{i_{n_i}}\} \subset \times_{i=2}^{d}[1, n]$ if each given d-dimensional hyper-rectangle intersects the set $[1, n] \times A$. Abusing notation slightly, we will say that a $(d-1)$-dimensional integer vector $(y_2, y_3, \ldots, y_d)$ is in A if $y_i \in \{a_{i_1}, a_{i_2}, \ldots, a_{i_{n_i}}\}$ for all i, and $|A| = \prod_{i=2}^{d} n_i$ will denote the maximum number of distinct integer vectors in A. We will show an *exact* algorithm for the A-restricted d-RPACK problem that runs in $O(p^{|A|+1}\, dk)$ time.

We first order the given hyper-rectangles in such a way that if $r \preceq r'$ then $s_1(r) \leq s_1(r')$, breaking ties arbitrarily. Consider a legal solution S of the d-RPACK problem, that is the hyper-rectangles in S are disjoint and $|S| \leq k$. We define the j^{th} element of S to be $r \in S$ such that $|\{r' \in S : r' \preceq r\}| = j$. Next, if r is the j^{th} element of S than the j^{th} cut of S is the following subset: $\{r' \in S : r' \preceq r \wedge t_1(r') \geq s_1(r)\}$. If $j = |S|$, then the j^{th} element of S is also called the *terminal element* of S. We define the *terminal cut* of S similarly.

Lemma 13 *If S is a legal solution with at least j hyper-rectangles, then the j^{th} cut of S has at most $|A|$ elements.*

Proof. Let us order the vectors in the set A. For a d-dimensional input hyper-rectangle r, the first $a \in A$ such that $r \cap ([1,n] \times \{a\}) \neq \emptyset$ will be denoted $\alpha(r)$. Because our input is A-restricted, $\alpha(r)$ is defined for every input hyper-rectangle r.

Now assume that r is the j^{th} element of S, $s = s_1(r)$ and T is the j^{th} cut of S. Consider $r' \in T$. Because $r' \preceq r$ we have $s_1(r') \leq s$. Because $r' \in T$, we have $t_1(r') \geq s$. Thus $s \in [s_1(r'), t_1(r')]$.

Now consider $r', r'' \in T$ such that $\alpha(r') = \alpha(r'') = a$. Then $s \times a \in r' \cap r''$. Because hyper-rectangles in T are disjoint, we can conclude that $r' = r''$. Thus α is a mapping that maps distinct elements of T to distinct elements (vectors) in A. Hence, $|T| \leq |A|$. ❑

Lemma 14 *Assume that S is a legal solution, $|S| < k$, r is the terminal element of S and $r \prec r'$. Then $S \cup \{r'\}$ is a legal solution if and only if r' does not intersect any rectangles from the terminal cut of S.*

Proof. Let T be the terminal cut of S. Obviously, if $S \cup \{r'\}$ is a legal solution then r' does not intersect T. Conversely, assume that r' does not intersect T. For $S \cup \{r'\}$ to be a legal solution, it suffices to show that r' does not intersect any hyper-rectangles from $S - T$. Because every hyper-rectangle $s \in S$ satisfies $s \preceq r$, $r'' \in S - T$ only if $t_1(r'') < s_1(r)$. Because $r \prec r'$, $s_1(r) \leq s_1(r')$. Thus $t_1(r'') < s_1(r')$, which implies that $r' \cap r'' = \emptyset$. ❑

Lemma 15 *There exists an exact algorithm for the A-restricted d-RPACK problem that runs in $O(p^{|A|+1}\ dk)$ time.*

Proof. We define a *plausible j^{th} cut of a legal solution* as a set of hyper-rectangles C such that $|C| \leq j$ and C is its own terminal cut. We also say that $\emptyset$ is a plausible 0^{th} cut of a legal solution. We define the following weighted directed acyclic graph. The *nodes* of the graph are all pairs (j, S) such that S is a plausible j^{th} cut of a legal solution for $j \in \{1, 2, \ldots, k\}$. If $p \leq |A|$ then $2^p = O(\frac{p^{|A|}}{|A|})$, otherwise $\binom{p}{|A|} < \frac{p^{|A|}}{|A|!} = O(\frac{p^{|A|}}{|A|})$. Hence, by Lemma 13, $|S| \leq |A|$ and we have at most $O(\frac{p^{|A|}}{|A|}\ k)$ nodes in this graph. An ordered pair $((j, S), (j+1, S'))$ forms a *directed edge* if it is plausible that for some legal solution S is the j^{th} cut and S' is the $(j+1)^{\text{st}}$ cut. More formally, S is a plausible j^{th} cut, for every $r \in S$ we have $r \prec r'$ and S' is the terminal cut of $S \cup \{r'\}$. The *weight* of this edge is equal to $w(r')$. Since there are at most $p-1$ choices for r', the out-degree of any node is less than p. Hence, the total number of edges in this graph is at most $O(\frac{p^{|A|+1}}{|A|}\ k)$. For any ordered pair $((j, S), (j+1, S'))$, whether there should be a directed edge between them can be checked in $O(d\ |A|)$ time. Hence, building this weighted graph takes at most $O(d\ |A|\ \frac{p^{|A|+1}}{|A|}\ k) = O(p^{|A|+1}\ dk)$ time. One can see that there is 1-1 correspondence between legal solutions and the directed paths in this graph that start at $(0, \emptyset)$. The optimum solution corresponds to the longest path. Because this is a directed acyclic graph, we can find the longest path in time proportional to the sum of the number of nodes and the number of edges in the graph [CLR90]. ❑

Our approximation algorithm for the d-RPACK problem has the following idea. We partition the input set into $(\lfloor 1 + \log_L n \rfloor)^{d-1}$ subsets, and then we find an *exact* solution for each subset.

We start from partitioning the range $[1, n]$ into $\lfloor 1 + \log_L n \rfloor$ subsets. For $j \geq 0$, we say that the j^{th} *lattice* is the set of integers in $\{1, 2, \ldots, n\}$ that are divisible by L^j but are not divisible by L^{j+1}.

Next, we say that an interval $[s, t]$ with $s, t \in \{1, 2, \ldots, n\}$ is of *class* j (denoted by $class(s, t) = j$) if it contains numbers from the j^{th} lattice, but it does not contain *any* number from $(j+1)^{\text{st}}$ lattice. We show that $class(s, t) \leq \log_L t$. If $class(s, t) = 0$, then obviously, $class(s, t) \leq \log_L t$. Otherwise, if $class(s, t) = j > 0$, then $L^j \leq t$, hence $j = class(s, t) \leq \log_L t$.

We partition our set of input hyper-rectangles into equivalence classes of the following relation $\bowtie$:

$$r \bowtie r' \text{ if an only if } class(s_i(r), t_i(r)) = class(s_i(r'), t_i(r')) \text{ for all } i \in \{2, 3, \ldots, d\}.$$

Because $t_i(r) \leq n$, there are at most $(\lfloor 1 + \log_L n \rfloor)^{d-1}$ equivalence classes of hyper-rectangles. The total time to partition the given set of p hyper-rectangles into these equivalence classes can be trivially done in $O(pd(\lfloor 1 + \log_L n \rfloor)^{d-1}) = O(d\ p^{(L-1)^{d-1}+1})$ time. It remains to show how to solve the d-RPACK problem for each class. Consider an equivalence class C such that $r \in C \equiv \bigwedge_{i=2}^{d} class(s_i(r), t_i(r)) = j_i$. First we find the connected components of C such that two hyper-rectangles r, r' belong to the same component if $r \cap r' \neq \emptyset$. The total time taken to find all these connected components for all (non-empty) equivalence classes is at most $O(dp^2)$. Observe for each connected component D, and $r, r' \in D$ the following holds for $i \in \{2, 3, 4, \ldots, d\}$

$$[s_i(r), t_i(r)] \cap [s_i(r'), t_i(r)] = \emptyset \equiv [s_i(r) \bmod L^{j_i+1}, t_i(r) \bmod L^{j_i+1}] \cap [s_i(r') \bmod L^{j_i+1}, t_i(r) \bmod L^{j_i+1}] = \emptyset$$

Therefore the d-RPACK problem on D remains same if we transform every hyper-rectangle using the **mod** operations as describe above, and then translate each component along the x_1 axis to assure that the hyper-rectangles from different components are still disjoint. More precisely, first for every $r = \times_{i=1}^{d}[s_i(r), t_i(r)] \in D$, we transform r to $r' = \times_{i=1}^{d}[s_i(r'), t_i(r')]$, where $s_1(r') = s_1(r)$, $t_1(r') = t_1(r)$, and $s_k(r') = s_k(r) \bmod L^{j_i+1}$ and $t_k(r') = t_k(r) \bmod L^{j_i+1}$ for $k \in \{2, 3, \ldots, d\}$. Then, if $D_1, D_2, \ldots, D_m$ are the connected components of C, then every transformed rectangle $r' = \times_{i=1}^{d}[s_i(r'), t_i(r')] \in D_i$ is replaced by $r'' = [s_1(r') + (i-1)n, t_1(r') + (i-1)n] \times (\times_{i=2}^{d}[s_i(r'), t_i(r')])$. If r_1 and r_2 are two original hyper-rectangles in C and r_1'' and r_2'' are these two hyper-rectangles after the above transformation, then it follows that $r_1 \cap r_2 = \emptyset \equiv r_1'' \cap r_2'' = \emptyset$.

We define $A_i = \{aL^{j_i} : a \in \{1, 2, 3, \ldots, L-1\}\}$ and let $A = \times_{i=2}^{d} A_i$. Obviously, $|A| = (L-1)^{d-1}$. After the last transformation each set $D \in C$ and hence C becomes A-restricted. Let $p_1, p_2, \ldots$ be the number of hyper-rectangles in different equivalence classes. The total time taken to solve the d-RPACK problem for all equivalence classes is then $O(dk \sum_i p_i^{(L-1)^{d-1}+1}) = O(p^{(L-1)^{d-1}+1}\ dk)$. ❑

5 Concluding Remarks

A general open issue is whether the approximation bounds in this paper can be further improved. It is known that it is NP-hard to approximate the RTILE problem to within an approximation ratio better than $5/4$ [KMP98], hence a gap still remains. Our algorithm for the RTILE problem for the $\{0, 1\}$-

arrays[9] in fact shows that there is a hierarchical partition of the given array in which the maximum weight of a tile is at most $2\lceil A/p \rceil$ since we used $\lceil A/p \rceil$ as a (trivial) lower bound on the optimum. It is easy to generate examples such that no tiling exists with the maximum tile of weight at most $5/3 \lceil W/p \rceil$. Hence, our approach for tiling on $\{0,1\}$-arrays will not yield better than $5/3$ approximation unless new lower bound techniques are used. We used one such new lower bound for the DRTILE problem in the proof of Theorem 3 (in Lemma 7) to get the improved approximation ratio of $2d-1$.

Initially, we suspected that $(\lfloor 1 + \log n \rfloor)^{d-1}$ is a lower bound on the approximation ratio that is attainable for the d-RPACK problem. However, to our surprise, we were able to obtain an approximation ratio which can be better than the logarithmic approximation bound by any arbitrary constant factor in Section 4.2. It remains open to understand the limits on the approximability of this problem, in particular, in $d > 1$ dimensions. Also, whether the ideas here will prove useful in applications domains remains to be seen.

References

[BNR96] V. Bafna, B. Narayanan and R. Ravi. Nonoverlapping local alignments (weighted independent sets of axis parallel rectangles). *Discrete Applied Mathematics*, 41–53, 1996.

[BD97] P. Berman and B. DasGupta. On the Complexities of Efficient Solutions of the Rectilinear Polygon Cover Problems. *Algorithmica*, 17, 331-356, 1997.

[CLR90] T. H. Cormen, C. E. Leiserson and R.L.Rivest. Introduction to Algorithms. The MIT Press, 1990.

[FM+96a] T. Fukuda, Y. Morimoto, S. Morishita and T. Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, Algorithms and Visualization. *Proc. ACM SIGMOD*, 1996.

[FM+96b] T. Fukuda, Y. Morimoto, S. Morishita and T. Tokuyama. Mining optimized association rules for numerical attributes. *Journal of Computer and Systems Sciences*, 58, 1–12, 1999.

[FPT81] R. Fowler, M. Paterson, and S. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Proc. Letters*, 12, 133–137, 1981.

[KMP98] S. Khanna, S. Muthukrishnan, and M. Paterson. Approximating rectangle tiling and packing. *Proc Symp. on Discrete Algorithms (SODA)*, pages 384–393, 1998.

[K80] R. P. Kooi. *The optimization of queries in relational databases.* PhD thesis, Case Western Reserve University, Sept 1980.

[LP00] K. Lorys and K. Paluch. Rectangle Tiling. *Third International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX 2000)*, Lecture Notes in Computer Science 1913, 206–213, Sept. 2000.

[9] The same effect happens with arbitrary arrays as well.

[MD88] M. Muralikrishna and David J Dewitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. *Proc. of ACM SIGMOD Conf*, pages 28–36, 1988.

[MPS99] S. Muthukrishnan, V. Poosala and T. Suel. On rectangular partitions in two dimensions: Algorithms, complexity and applications. *Intl. Conf. Database Theory* (ICDT), 236-256, 1999.

[O88] M. H Overmars. Computational geometry on a grid: an overview, Theoretical Foundations of Computer Graphics and CAD. NATO ASI Series F40, Edited by R. A. Earnshaw, Springer-Verlag Berlin Heidelberg, 167-184, 1988.

[P97] V. Poosala. *Histogram-based estimation techniques in databases.* PhD thesis, Univ. of Wisconsin-Madison, 1997.

[RS99] R. Rastogi and K. Shim Mining optimized support rules for numerical attributes. *Proc. Intl Conf. Data Engineering*, 1999.

[S99] J. P. Sharp. Tiling Multi-Dimensional Arrays. *Foundations of Computing Theory*, 1999.

[SS99] A. Smith and S. Suri. Rectangular tiling in multi-dimensional arrays. *Proc. ACM-SIAM Symp on Discrete Algorithms* (SODA), 1999.

APPENDIX (proofs of Lemmas 2, 3, 4 and 5)

Proof of Lemma 2. By induction on b. If $b = 1$, then $T_j \leq 11$ and the claim is obvious. Otherwise, find the *longest* initial 11-good part (prefix) of row $\mathbf{T}_j$. Note that the weight of this part exceeds 6, otherwise we would extend it by one more element, thereby increasing its weight to at most $6+5$. The remaining part of $\mathbf{T}_j$ has weight at most $(b-1)6+5$ and by the inductive hypothesis we can partition it into $b-1$ 11-good rectangles. ❑

Proof of Lemma 3. By induction on b. If $b = 2$, find the longest initial 11-good part (prefix) of $\mathbf{T}_j$. If its weight is at least $6+y$, then the remaining part of $\mathbf{T}_j$ has weight at most $2 \times 6+5+y-(6+y) = 11$, and we succeeded in partitioning $\mathbf{T}_j$ into 2 11-good rectangles. If this weight is lower than $6+y$, try the same with the longest final 11-good part (suffix) of $\mathbf{T}_j$. If we fail both times, then both 11-good parts of $\mathbf{T}_j$ that we obtained have weights between 6 and $6+y$. Consequently, the remaining middle part has a weight of at most $5+y$.

For the inductive step, find the longest initial 11-good part (prefix) of $\mathbf{T}_j$. If its weights is at least $6+y$, then the remaining part of $\mathbf{T}_j$ (of total weight at most $6(b-1)+5$) can be partitioned into $b-1$ good parts by Lemma 2. Otherwise, its weight is between 6 and $6+y$ and the weight of the remaining part of $\mathbf{T}_j$ is at most $6(b-1)+5+y$, so it can be partitioned in the desired fashion by the inductive hypothesis. ❑

Proof of Lemma 4. Let $S_j = 6a-2+x$ for some integer $a \geq 3$ and a real x, $0 \leq x < 6$. It is sufficient to partition $\mathbf{S}_j$ into a 11-good rectangles, since then the deficit is $d_j = 5a-(6a-2+x) = 2-a-x \leq -1$.

If $T_j \le 6a-1$ then for $b = a-1$ we have $T_j \le 6(b+1)-1 = 6b+5$. Using Lemma 2 we can partition $\mathbf{T}_j$ into $a-1$ 11-good rectangles, and $\mathbf{B}_j$ provides the a^{th} 11-good rectangle.

Otherwise, for some real y $(0 < y < 5)$, $T_j = 6a-1+y$, and $B_j = S_j - T_j = 6a-2+x-(6a-1+y) = x-y-1$. For $b = a-1$ we have $T_j = 6b+5+y$, thus we can partition $\mathbf{T}_j$ according to Lemma 3. If this partition contains $a-1$ good rectangles, we can add $\mathbf{B}_j$ as the a^{th} good rectangle. If this partition contains a rectangles, each with weight at most $6+y$, then we can extend these rectangles vertically to cover $\mathbf{B}_j$. The weight of an extended rectangle is at most $6+y+B_j = 6+y+x-y-1 = 5+x < 11$, hence each extended rectangle is 11-good. ❑

Proof of Lemma 5. We partition $\mathbf{S}_j$ into $\mathbf{T}_j$ and $\mathbf{B}_j$. $\mathbf{B}_j$ is always 11-good, and $\mathbf{T}_j$ is good by the assumption. The deficit is $d_j = 10 - S_j < -1$. ❑