Slice and Dice : A Simple, Improved Approximate Tiling Recipe

Piotr Berman* Bhaskar DasGupta† S. Muthukrishnan‡

Abstract

We are given a two dimensional array $\mathbf{A}[1 \cdots n, 1 \cdots n]$ where each $\mathbf{A}[i, j]$ stores a non-negative number. A (rectangular) tiling of $\mathbf{A}$ is a collection of rectangular portions $\mathbf{A}[l \cdots r, t \cdots b]$, called *tiles*, such that no two tiles overlap and each entry $\mathbf{A}[i, j]$ is contained in a tile. The *weight* of a tile is the *sum* of all array entries in it.

In the `MAX-MIN` problem, we are given a weight bound W and our goal is to find a tiling such that (a) each tile is of weight *at least* W (the `MIN` condition) and (b) the number of tiles is *maximized* (the `MAX` condition). In the `MIN-MAX` problem, we are given a weight bound W again and our goal is to find a tiling such that (a) each tile has weight *at most* W and (b) the number of tiles is *minimized*. These two basic problems have many variations depending on the weight functions, whether some areas of $\mathbf{A}$ must not be covered, or whether some portion of $\mathbf{A}$ may be discarded, etc. These problems are not only natural combinatorial problems, but also arise in a plethora of applications, e.g., in databases and data mining, video compression, load balancing, building index structures, manufacturing and so forth.

Both the above tiling problems (as well as all of their variations relevant to this paper) are known to be NP-hard. In this paper, we present approximations algorithms for solving these problems based on *epicurean* methods : variations of a basic *slice-and-dice* technique. Surprisingly, these simple algorithms yield small constant factor approximations for all these problems. For some of the problems, our results are the first known approximations; for others, our results improve the known algorithms significantly in approximation bounds and/or running time. Of independent interest are the tight bounds we show for sizes of the binary space partition trees for isothetic rectangles.

1 Introduction

The problems considered in this paper resemble cutting a pie, albeit with some idiosyncrasies. The pieces we cut are rectangles, not wedges or squares, so each piece resembles a Sicilian Pie. The pie itself is rectilinear in shape. Furthermore, the pie is not uniformly done : there are portions of the pie we need to avoid, some we do not care about, and some we may carve out of consideration. As an epicurean instance, this may be far-fetched but the formal problems we study of this flavor are quite natural and arise in a plethora of applications.

The problems we study concern tiling a rectilinear region to optimize a `MAX/MIN` or `MIN/MAX` criteria. These problems are well-motivated from a number of applications in, for example, databases and data mining, video compression and manufacturing. Some of these problems have a rich history while the others are novel. All of them are NP-hard[1], so our focus is on providing approximation results. Our main results are efficient improved algorithms for these problems with small constant factor approximations. We obtain these results using variations of simple slicing and dicing of the rectilinear region. In what follows, we present our problems and results formally.

1.1 Formal Statement of Problems

We start with two basic problems to focus this discussion. We are given a two dimensional array $\mathbf{A}[1 \cdots n, 1 \cdots n]$ of size $N = n^2$ where each $\mathbf{A}[i, j]$ stores a non-negative number. A (rectangular) tiling of $\mathbf{A}$ is a collection of rectangular portions $\mathbf{A}[l \cdots r, t \cdots b]$, called *tiles*, such that no two tiles overlap and each entry $\mathbf{A}[i, j]$ is contained in a tile. The *weight* of a tile is the *sum* of all array entries in it[2].

In the `MAX-MIN` problem, we are given a weight bound W and our goal is to find a tiling such that (a) each tile is of weight *at least* W (the `MIN` condition) and (b) the number of tiles is *maximized* (the `MAX` condition). Obviously, there is a feasible solution if and only if sum of all elements of $\mathbf{A}$ is at least W, the entire array being a trivial tile. In the `MIN-MAX` problem, we are again given a weight bound W and our goal is to find a tiling

*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 NLM grant LM05110.

†Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607. Email: `dasgupta@cs.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`

[1] In contrast, if one were to cut a french bread, i.e., one dimensional array, even under similar conditions we have outlined above, it corresponds to one dimensional versions of our problems all of which are solvable in polynomial time by dynamic programming or greedy methods.

[2] Unless otherwise stated, in our description of the tiling problems we will 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.

such that (a) each tile has weight *at most* W, and (b) the number of tiles is *minimized*. Obviously, there is a feasible solution if and only if each array item is at most W, the collection of each array element being a trivial tiling.

Both versions of the basic problem are natural and have good motivations. However, to our knowledge, the `MAX-MIN` version has not been studied in literature before[3], while `MIN-MAX` problem has been studied many times within the Algorithms community [5, 17, 20, 24, 33, 34] (see [20] for some background). Later on, our main focus in considering applications of these problems will lead to variations of these tiling problems where, in general, the weight of a tile is a *more complicated function* of the tile elements. Furthermore, there are novel variations of these problems where (a) array **A** may have *holes* (rectilinear regions) that should not be covered, or (b) a given number of tiles of **A** may be removed, etc. We will show how to modify our techniques for solving the two basic problems stated above to address *all* these variations. We will discuss those variations and applications in Section 6 and focus only on the two basic problems for most part.

1.2 Our Results We use variations of what we informally refer to as "slice-and-dice" technique (or simply slicing and dicing) to obtain efficient approximation algorithms for the abovementioned tiling problems. More specifically, our main results are *threefold*:

1. *(Greedy Slice-and-Dice)* We present the first known approximation algorithm for the `MAX-MIN` problem: it uses at least $\left(\frac{1}{3}\right)^{\text{rd}}$ of the maximum number of rectangles each of weight at least W; if the array elements can assume only binary values, which is an important special case, this ratio $\frac{1}{3}$ is improved to $\frac{2}{5}$. The algorithm is based on *greedily* slicing the array into strips and dicing each slice; the running time is *linear* (in fact, linear in the number of non-zero elements of the given array) and is very simple to implement. The technical crux of this result in the analysis of the slice and dice technique.

   Although the `MAX-MIN` problem appears to be closely related to the `MIN-MAX` version, there are fundamental differences. In particular, if we take a feasible solution for the `MIN-MAX` problem and further divide some of the tiles, the solution remains a feasible solution; hence a slice that is different from the optimal one may be "fixed" with a few additional slices. In contrast, this property *does not hold* for the `MAX-MIN` problem, and as a result it is crucial that the slices be almost perfect.[4] Our greedy slice-and-dice algorithm is similar in flavor to that for the `MIN-MAX` problem from [5], however, details, analysis and lower bounds used are all quite different. It is very surprising that a simple greedy slice-and-dice algorithm works well for the `MAX-MIN` version of the tiling problem.

2. *(Recursive Slice-and-Dice: Binary Space Partitions)* A recursive application of the slice-and-dice solution to a tiling problem can be thought as a Binary Space Partition (BSP) of the tiles (rectangles). Therefore, a general approach to solving the tiling problems is to use BSPs of isothetic rectangles, wherein the size of the BSP affects the quality of approximation of our solution. (See Section 4.2 for the definition and further details; BSPs for various objects are of significant independent interest in general.) For the purpose of our applications, it is sufficient to consider a special type of BSP, commonly called *Binary Space Auto-partition*, in which every cut is either a *horizontal* or a *vertical* line.

   We show that given n isothetic rectangles, there exists a binary space auto-partition tree (and, hence a BSP tree) of size at most $3n-2$. If the given set of rectangles partition their smallest rectangular bounding box (which is of interest to our tiling applications), then we prove an improved upper bound of $2n-1$.

   Paterson and Yao proved an upper bound of $12n$ in [29] in their seminal paper; subsequent improvements have led to the previous best upper bound of $4n$ [13, 8]. Our result above improves this by a factor of $4/3$ in general, and a factor of 2 in the tiling case. Lately, there has been a great interest in BSPs and their applications in general [1, 10, 11] and in BSPs for rectangles in particular [11]. Since we proved this upper bound for the tiling case in a preliminary writeup [4], Dumitrescu et al. [11] proved a lower bound of $2n - o(n)$, matching our upper bound within a lower-order term[5].

   We use our BSP results in two ways for tiling problems:

   (a) We present a *bicriteria* approximation algorithm for the `MAX-MIN` problem by which we

[3] This problem came up in a personal communication [30].

[4] A feasible solution for the `MAX-MIN` problem will remain feasible if we combine two or more tiles into one, but this is a difficult operation to coordinate because the tiles to be merge need to be *aligned*. In contrast, further slicing of a tile is a simple local operation which helps in the `MIN-MAX` problem.

[5] Dumitrescu et al. [11] proved a lower bound of $2n - o(n)$ on the size of auto-partition trees of n line segments each of which is either horizontal or vertical.

produce a tiling with at least as many tiles as in an optimum solution and still guarantee that the tiling contains a collection of at least $\frac{1}{2}$ of the optimum number of tiles each having weight at least $\frac{1}{4}W$. This is the first result that produces no fewer tiles than the optimum while still partly meeting the minimum weight criteria[6].

(b) We improve running times of approximation algorithms for the `MIN-MAX` problem using the BSP approach. The previously best known approximation algorithm had a performance ratio of 2 but ran in $N^{\geq 10}$ time. Our improved running time (with the same approximation ratio) is $O(N^{\frac{5}{2}})$ which is more manageable.

The advantage of our BSP based approach is that it is a general technique applicable to not only the basic `MIN-MAX` and `MAX-MIN` problems described before but also to their variations with different weight functions, with holes, when some parts of the array may be removed, etc. We will briefly comment on this later.

3. *(Slice-and-Dice with Dynamic Programming)* We consider the `MIN-MAX` problem where the tiles are nearly, but not exactly, uniform (after all, even sloppy Sicilian pies may not be too far apart from each other in size!). Say that the *global* aspect ratio, namely, the ratio of largest side of any tile to the smallest of any tile is bounded by a constant. We present an $O(N/\delta^5)$ time algorithm that returns an $2+\delta$ approximation for this problem, for any $\delta > 0$. It follows from previously known results that even this version of the problem is NP-hard. Previous algorithms for this problem returned a 2-approximation in $O(N^{>5})$ time or a 4 approximation in $O(N^{\frac{5}{2}})$ time[7]. Here, our techniques involve dynamic programming (as is common in designing PTAS) to pick the appropriate slice-and-dice combinations.

   We further extend this result to provide a PTAS, that is, polynomial time $(1+\varepsilon)$-approximation for any constant $\varepsilon > 0$. However, the running time of the resulting algorithm, while being polynomial in N, is too prohibitive to be practical for small values of ε. This result however is of theoretical significance: the "dual" of this problem — namely that given an upper bound B on the number of tiles, minimize the maximum weight of the tiles — cannot be approximated to better than a factor of $\frac{5}{4}$ (see [17]). Hence this problem is provably simpler than its dual, the *first* such instance known for any rectangular tiling problem.

As mentioned above, we use the slice-and-dice technique in a unified way to solve the tiling problems. In general, an *informal* description of this culinary technique consists of the following steps:

- We *slice* the array, that is, partition the input array into a number of slices (rectangles) satisfying certain optimization criterion depending on the problem. Such a partitioning scheme can be obtained by either greedy slicing (Section 3), binary space partitions (Section 4.3) or dynamic programming on slices generated using shifting technique (Section 5).
- Depending on the problem, we may need to adjust the slices locally. A local adjustment or *dicing* step may typically consist of looking at a few (typically a small constant) number of nearby slices and repartitioning the entries of the input array spanned by them to obtain satisfactory approximation results.

Some previous results have used variations or specific implementations of the slice-and-dice techniques to obtain approximation algorithms for specific tiling problems [17, 33, 34, 5]. Our paper uses it as a unified framework to obtain improved approximation algorithms for the `MIN-MAX` tiling problems as well as the obtaining the first nontrivial approximation algorithms for the `MAX-MIN` tiling problems. Why the slice-and-dice technique by itself is conceptually and algorithmically simple, the crux of our technical work is in the analyses. We expect slice-and-dice to be useful elsewhere in tiling problems in the future.

1.3 The Map We present our approximation algorithms for the `MAX-MIN` problem in Section 3 using the greedy slice-and-dice technique, our approximation algorithms for the `MIN-MAX` and `MAX-MIN` problems using generalized slice-and-dice (involving BSPs) in Section 4.3 and our approximation algorithms for the general `MIN-MAX` tiling problems with shifted slice-and-dice technique in Section 5. In Section 6, we present a selection of applications of our results. In Section 7 we present some concluding remarks with possible future research directions. Due to space limitation, many

[6] Tradeoff between the constants $\frac{1}{2}$ and $\frac{1}{4}$ is also possible; see Theorem 4.4 for details.

[7] There has been a considerable amount of work of late in proving efficient solutions for geometric problems where the objects are uniform in some *local* sense only (e.g., see [1, 9]). However, it is easy to see that our technique will not provide any better solutions if we instead assume that the aspect ratios of individual tiles are bounded.

proofs are omitted; they are available in the full version of the paper.

2 Basic Definitions and Notations

For a set of rectangles $R_1, R_2, \ldots, R_n$, with $R_i = [a_i, b_i] \times [c_i, d_i]$ where $\times$ denotes the Cartesian product, let the *global aspect ratio* be defined as $\frac{\max_{1\le i\le n}\{(b_i - a_i), (d_i - c_i)\}}{\min_{1\le i\le n}\{(b_i - a_i), (d_i - c_i)\}}$. A rectilinear polygon is a simple polygon with its sides parallel to the coordinate axes; such a polygon may or may not have holes but if the holes are present then they are also rectilinear (degenerate (point) holes are allowed). An array $\mathbf{A}$ is called a *binary* array if all of its entries are either 0 or 1, otherwise it is called an *arbitrary* array; unless otherwise stated, an array is an arbitrary array. A polynomial-time approximation scheme (PTAS) for a minimization problem is an algorithm that takes as input an instance of the problem of size n and a constant $\varepsilon > 0$ and produces a solution whose value is at most $(1 + \varepsilon)$ times that of the optimum solution in time polynomial in n.

3 MAX-MIN Problem Via Greedy Slicing and Dicing

In this section, we consider MAX-MIN tiling problem in which the input is a two dimensional array $\mathbf{A}$ of size $n \times n$ containing non-negative numbers and the weight of a tile (subarray) is the sum of all elements of $\mathbf{A}$ that fall inside it. If $\mathbf{A}$ is sparse, containing m non-zero entries ($n \le m \le N = n^2$), 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. We assume that our input arrays are represented this way. The MAX-MIN tiling problem is defined as follows.

MAX-MIN tiling problem. *Given a two dimensional array $\mathbf{A}$ of size $n \times n$ containing non-negative numbers and a positive integer W, partition $\mathbf{A}$ (if possible) into a maximum number of tiles so that the minimum weight of any tile is at least W.*

We may assume that $W = 1$ by scaling all the entries of $\mathbf{A}$, if necessary. Now, a *feasible* solution for the MAX-MIN problem is a tiling of $\mathbf{A}$ in which every tile has a weight of at least 1. Obviously, we may assume that $A \ge 1$, since otherwise the MAX-MIN problem has no feasible solution. Let $b = \max_{1\le i,j\le n} \mathbf{A}[i, j]$. We may assume that $b \le 1$ by using the following crucial observation.

OBSERVATION 1. *Given an instance array $\mathbf{A}$ of the MAX-MIN problem, let $\mathbf{A}'$ be the array obtained from $\mathbf{A}$ in which every element larger than 1 is replaced by 1. Then, any feasible solution for $\mathbf{A}$ is also a feasible solution for $\mathbf{A}'$ and vice versa.*

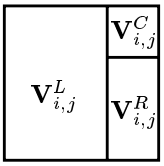
The main result of this section is as follows.

THEOREM 3.1. *There exists an approximation algorithm for the MAX-MIN tiling problem that runs in $O(n + m)$ time and produces a tiling using at least $\frac{1}{r}(p^* - 2)$ tiles, where p^* is the (maximum) number of tiles used by an optimal algorithm and* $r = \begin{cases} 5/2 & \text{if } \mathbf{A} \text{ is binary} \\ 3 & \text{otherwise} \end{cases}$

Proof. First, we describe a basic *slicing* algorithm that is used by the algorithm. The slicing algorithm partitions the input array $\mathbf{A}$ into *slices*; this algorithm is used as a routine in our approximation algorithm. A slice is a tile that consists of complete rows. The slicing algorithms starts from the bottom and proceeds upwards, finding minimal slices that have weight at least 1; the last slice (possibly empty) may have a weight less than 1 and is called a *remainder slice*. More precisely, the algorithm computes l and array entries $t[0] = 0, t[1], \ldots, t[l]$, so that slice $\mathbf{S}_i$ consists of rows $\mathbf{R}_{t[i-1]+1}$ to $\mathbf{R}_{t[i]}$, while the topmost remainder slice starts at row $\mathbf{R}_{t[l]+1}$. It is easy to see that the slicing algorithm can be implemented in $O(n + m)$ time.

Define a tile to be *good* if its total weight is at least 1. We present an algorithm that always finds a solution with t good tiles such that $A < rt + 2$. We begin our algorithm by slicing $\mathbf{A}$ using our slicing algorithms as described above. Our dicing step will then consist of partitioning the union of non-remainder slices into good tiles and covering the remainder slice with the extensions of adjacent tiles of that slice.

We now describe our dicing step more precisely. First, we consider the case when $\mathbf{A}$ is arbitrary. We partition each slice $\mathbf{S}_i$ using the same slicing algorithm as described before, except that we consider columns of $\mathbf{S}_i$ rather than the rows. This produces vertical slices, *V-slices* for short. We denote the number of regular V-slices obtained from $\mathbf{S}_i$ with $l(i)$, so the V-slices $\mathbf{V}_{i,1}, \ldots \mathbf{V}_{i,l(i)}$ are regular and $\mathbf{V}_{i,l(i)+1}$ is the *remainder V-slice*. To obtain our preliminary partition, we combine each remainder V-slice with its preceeding regular V-slice.



Later, we split each regular V-slice $\mathbf{V}_{i,j}$ into three parts:

$\mathbf{V}^L_{i,j}$ that consists of all columns except the last

$\mathbf{V}^R_{i,j}$ that consists of the last column except its top

$\mathbf{V}^C_{i,j}$ that consists of the the top of last column

We can estimate the weights of these parts as follows:

because $\mathbf{V}^L_{i,j}$ did not make a complete V-slice, $V^L_{i,j} < 1$;
because $\mathbf{V}^R_{i,j}$ together did not make a complete slice, $\sum_{j=1}^{l(i)} V^R_{i,j} < 1$;
because no entry exceeds 1, $V^C_{i,j} \le 1$.

Consequently, $S_i = V_{l(i)+1} + \sum_{j=1}^{l(i)}(V^L_{i,j} + V^R_{i,j} + V^C_{i,j}) < 2 + 2l(i)$.

Before we continue, we can observe that we can already guarantee to get at least $\lfloor A \rfloor/4$ tiles. Note that $S_i < 2 + 2l(i) \le 4l(i)$ and $A < 1 + \sum_{i=1}^{l} S_i$, while we got $t = \sum_{i=1}^{l} l(i)$ tiles. Therefore $4t > A - 1$ and $4l \ge \lfloor A \rfloor$. Thus at the moment we have an algorithm with approximation ratio 4.

To improve this ratio, we do the following additional dicing:

for $i \leftarrow 1$ **to** $l-1$ **do**
  if slice $\mathbf{S}_i$ was not modified,
    $l(i) = 1$ and $l(i+1) \le 2$ **then**
      if possible, partition $\mathbf{S}_i \cup \mathbf{S}_{i+1}$
      into $l(i) + l(i+1) + 1$ good tiles

To show that we we will obtain a number t of good tiles such that $t \ge \lfloor (A+1)/3 \rfloor$, it suffices to show that $A < 3t + 2$. This in turn will follow from $\sum_{i=1}^{l} S_i < 3t + 1$.

We define $\sigma(i)$ to be 1 if our final **for** loop increased the number of tiles while processing $\mathbf{S}_{i-1} \cup \mathbf{S}_i$, and 0 otherwise. Clearly, our algorithm produces $t = \sum_{i=1}^{l} l(i) + \sigma(i)$ tiles, so we need to show that $\sum_{i=1}^{l}[S_i - 3l(i) - 3\sigma(i)] < 1$.

We do it by induction on l. Our inductive claim is the following:

if $\zeta(k) = \sum_{i=1}^{k}[S_i - 3l(i) - 3\sigma(i)] > 0$, **then**
  $l(k) = 1$, $S_i - V^L_{k,1} > 1$ and
  $\zeta(k) < V^L_{k,1} + V_{k,2} - 1$

Recall that both $V^L_{k,1}$ and $V_{k,2}$ are smaller than 1, so our inductive claim implies $\zeta(k) < 1$. We can put $\zeta(0) = 0$ and the inductive basis is trivial. For the inductive step, we assume that the claim holds for $\zeta(k-1)$ and then we consider several cases.

Case: $l(k) > 2$, then $\zeta(k) < \zeta(k-1) + S_k - 3l(k) < 1 + 2 + 2l(i) - 3l(i) = 3 - l(i) \le 0$.

Case: $\zeta(k-1) \le 0$ and $l(k) = 2$, then $\zeta(k) < S_k - 3l(k) < 2 + 4 - 6$.

Case: $\zeta(k-1) \le 0$ and $l(k) = 1$, then $\zeta(k) < S_k - 3l(k) < V^L_{k,1} + V^C_{k,1} + V^R_{k,1} + V_{k,2} - 3 < V^L_{k,1} + V_{k,2} - 1$.

Case: $\sigma(k) = 1$, then $\zeta(k) < \zeta(k-1) + S_k - 3l(k) - 3 < 1 + 2 + 2l(i) - 3l(i) - 3 = -l(i)$.

In further cases we can assume that $\zeta(k-1) > 0$, $\sigma(k) = 0$ and $l(k)$ equals 1 or 2. First we consider cases for $l(k) = 1$. We will use the following notation: $\mathbf{V}^L_{k-1,1} = \mathbf{C}$, $\mathbf{V}^C_{k-1,1} \cup \mathbf{V}^R_{k-1,1} = \mathbf{D}$, $\mathbf{V}_{k-1,2} = \mathbf{E}$, $\mathbf{V}^L_{k,1} = \mathbf{C}'$, $\mathbf{V}^C_{k,1} \cup \mathbf{V}^R_{k,1} = \mathbf{D}'$, $\mathbf{V}_{k,2} = \mathbf{E}'$. The inductive hypothesis is $\zeta(k-1) < C + E - 1$ and $S_{k-1} - C > 1$.

Case: $S_k - C' < 1$, then $S_k < 2$ and $\zeta(k) < 1 + 2 - 3$.

Case: $\mathbf{D}$ and $\mathbf{D}'$ are in the same column. One attempt to partition $\mathbf{S}_{k-1} \cup \mathbf{S}_k$ is $\mathbf{C} \cup \mathbf{C}'$, $\mathbf{S}_{k-1} - \mathbf{C}$, and $\mathbf{S}_k - \mathbf{C}'$. The latter two tiles are good and since $\sigma(k) = 0$ we have $C + C' < 1$. For a symmetric reason $E + E' < 1$. Thus $\zeta(k) < C + E - 1 + C' + D' + E' - 3 < C' - 2 < 0$.

For the further cases we use notation $\mathbf{V}^C_{k,1} = \mathcal{D}'_u$ and $\mathbf{V}^R_{k,1} = D'_d$.

Case: $\mathbf{D}'$ is in a column of $\mathbf{C}$. One attempt to partition $\mathbf{S}_{k-1} + \mathbf{S}_k$ is to extend $\mathbf{C}$ upward to cover $\mathbf{D}'_d$, use the top row of $\mathbf{S}_k$ and the rest. The latter two tiles must be good (if the top row of $\mathbf{S}_k$ is not good, $S_k < 2$ and we covered that before). Because the first tile is not good, we have $C + D'_d < 1$, and thus $\zeta(k) < C + E - 1 + C' + D'_d + D'_u + E' - 3 < E + C' + D'_u + E' - 3 < C' + E' - 1$.

Case: $\mathbf{D}'$ is in a column of $\mathbf{E}$. Symmetric to the last case.

In the remaining cases we assume that $l(k) = 2$, so we have to show $\zeta(k) < 0$. We use the same notation as above, except that now $\mathbf{V}^R_{l,2} = \mathbf{F}'_d$.

Case: $S_k < 5$; same as $l(k) = 1$ and $S_k < 2$.

Case: $\mathbf{D}$ and $\mathbf{D}'$ are in the same column. One attempt to partition $\mathbf{S}_{k-1} \cup \mathbf{S}_k$ is $\mathbf{C} \cup \mathbf{C}'$, $\mathbf{S}_{k-1} - \mathbf{C}$, and two tiles made of $\mathbf{S}_k - C'$. If we cannot make two tiles of $\mathbf{S}_k - C'$ then $S_k - C' < 4$ and thus $S_k < 5$, a case already covered. Thus this attempt fails only because $C + C' < 1$. Note also that $S_k - C' < 5$, thus $\zeta(k) <$

$C + E - 1 + S_k - 6 = (C + C') + E - 1 + (S_k - C') - 6 < 1 + 1 - 1 + 5 - 6 = 0$.

Case: $\mathbf{D}$ and $\mathbf{F}'_d$ are in the same column. Symmetric to the last case.

Case: whatever remains. Now neither $\mathbf{D}'$ nor $\mathbf{F}'$ is in the same column as $\mathbf{D}$. To make analysis more succinct, we introduce a bit more notation. First, $X = S_k - D'_d - E'_d$. By our estimates on the weights of pieces of vertical slices, $X < 5$. Second, Y is the sum of weights of those among $\mathbf{D}'_d$ and $\mathbf{F}'_d$ that are in the columns of $\mathbf{C}$, and Z of those that are in the columns of $\mathbf{E}$. In our case we have $S_k = X + Y + Z$.

One attempt to partition $\mathbf{S}_{k-1} + \mathbf{S}_k$ is to extend $\mathbf{C}$ upward to cover $\mathbf{D}'_d$, use the top row of $\mathbf{S}_k$ to form two tiles and the rest. Because this attempt failed, $C + Y < 1$. Similarly, $E + Z < 1$. Thus $\zeta(k) < C + E - 1 + S_k - 6 = (C + Y) + (E + Z) + X - 7 < 0$. It is easy to implement the dicing part of our algorithm in $O(n + m)$ time, so the overall time is still $O(n + m)$.

This ends the analysis of the algorithm for the case when $\mathbf{A}$ is arbitrary. If $\mathbf{A}$ was binary, then after rescaling the entries such that $W = 1$, we may assume that each non-zero element of $\mathbf{A}$ is equal to $\frac{1}{W}$ where W is an integer. We will prove that $A < 2.5t + 2$. The proof that $A < 2.5t + 1.5$ is similar to the previous, so we will list the differences.

1. $V^R_{i,j}, V_{i,l(i)+1} \leq 1 - 1/W$, $V^C_{i,j} \leq 1/W$, $\sum_{j=1}^{l(i)} V^R_{i,j} \leq 1 - 1/W$, thus $S_i \leq l(i) + 2 - 2/W$.

2. We will show that the number t of tiles produced by our algorithm satisfies $A < 5/2 + 3/2$ by showing that $\zeta(k) = \sum_{i=1}^{k} [S_i - 5/2\, l(i) - 5/2\, \sigma(i)] < 1/2$.

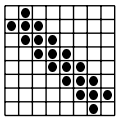
3. Our inductive claim is: if $\zeta(k) > 0$ then $l(k) = 1$ and $\zeta(k) \leq \S_k - 5/2$. Note that this means that $\zeta(k) \leq 1/2 - 2/W$.

4. The basis is as trivial as before. New case analysis for the inductive step:
Case: $l(k) > 1$ then $\zeta(k) < 1/2 + S_k - 5/2 < l(i) + 2 - 5/2\, l(i) < 2 - l(i) \leq 0$.

Case: $l(k) = 1$ and $\zeta(k-1) \leq 0$, the claim is obvious.

Case: Remaining case, $l(k) = 1$ and $\zeta(k-1) > 0$, hence $l(k-1) = 1$ and $\zeta(k-1) \leq S_{k-1} - 3/2$. Suppose that $\zeta(k-1) + S_k - 3/2 > 0$, then $S_{k-1} + S_k > 5$. Because the weight of all the rows of a slice, except the top, is at most $1 - 1/W$, this means that the sum of weights of the top rows of $\mathbf{S}_{k-1}$ and $\mathbf{S}_k$ is at least $3 + 2/W$. Apply the vertical slicing algorithm to $\mathbf{S}_{k-1} \cup \mathbf{S}_k$. The last column in a vertical slice has at most 2 non-zeroes from the top rows, and the previous columns have at most $W - 1$ non-zeroes, thus the weight of the intersection of a vertical slice with the two top rows is at most $1 + 1/W$. Thus after creating the first 2 vertical slices we still have weight 1 available in the two top rows. Consequently, we get a success of the attempt to get more tiles, $\sigma(k) = 1$ and $\zeta(k) \leq S_{k-1} + S_k - 15/2 < 0$.

Remark: In the proof of the above theorem, we used the total weight A of the input array $\mathbf{A}$ as an (obvious) upper bound on the number of tiles in an optimum solution. The following example shows that an alternative lower bound is necessary to prove better performance ratios for arbitrary arrays. For every $t > 0$, we can construct a corresponding array $\mathbf{A}$ such that t is the maximum number of good tiles in the partition and $\lceil A \rceil = 3t + 2$. Our $\mathbf{A}$ is a $(t + 2) \times (t + 2)$ array where every non-zero entry equals $1 - \frac{1}{4t}$; there are $3t+2$ non-zeros distributed in three diagonals in a manner shown in the adjacent diagram (where $\bullet$ indicates a non-zero). One can see that a good tile must contain at least two non-zeros. A brief inspection shows that every good tile must contain a non-zero from the central diagonal. Consequently, there cannot be more than t disjoint good tiles.



4 Recursive Slice-and-Dice: Binary Space Partitions

In this section, we will define general versions of the `MAX-MIN` and `MIN-MAX` tiling problems. We will also prove improved upper bounds for the size of Binary Space Partitions for a set of isothetic rectangles. Finally, we will show how we can obtain improved approximation algorithms for the general tiling problems using the bounds on the BSP size and a recursive version of the slice-and-dice technique.

4.1 General Versions of Tiling Problems

First, we define a *general* version of the `MIN-MAX` problem as follows. We are given an $n \times n$ array $\mathbf{A}$ of $N = n^2$ positive numbers, a parameter $W > 0$ and a weight function f mapping any subarray $\mathbf{R}$ of $\mathbf{A}$ to a positive real number such that f is *non-decreasing* [24], that is, for any two rectangles $\mathbf{T}_1$ and $\mathbf{T}_2$ forming a partition of a tile $\mathbf{T}$ of $\mathbf{A}$, $f(\mathbf{T}) \geq \max\{f(\mathbf{T}_1), f(\mathbf{T}_2)\}$[8]. The goal of

[8] The basic `MIN-MAX` problem is simply a *special case* of this problem when the weight of any tile is the sum of the entries in it.

the general MIN-MAX problem is to partition $\mathbf{A}$ with the minimum number tiles such that the maximum weight of any tile is at most W. Some of the commonly used weight functions are the sum squared error of each entry from the average of the entries in that tile, or simply the maximum of all entries in the tile; see [24] for details. Assume that, given the values of $f(\mathbf{T}_1)$ and $f(\mathbf{T}_2)$ for a partition $\mathbf{T}_1$ and $\mathbf{T}_2$ of a tile $\mathbf{T}$ of $\mathbf{A}$, we can compute $f(\mathbf{T})$ in time α. For example, $\alpha = O(1)$ if the weight function is the maximum of all entries in the tile or for the MIN-MAX problem.

A *general* version of the MAX-MIN problem can also be defined in a similar manner. The weight function f is again non-decreasing, but the goal now is to partition $\mathbf{A}$ with the maximum number of tiles such that the weight of each tile is at least W.

4.2 BSP: Definitions and Bounds Given a rectangular region $\mathbf{R}$ containing a set of n disjoint isothetic rectangles, a BSP of $\mathbf{R}$ consists of recursively partitioning $\mathbf{R}$ by a horizontal or vertical line into two subregions and continuing in this manner for each of the two subregions until each obtained region contains at most one rectangle. If a rectangle is intersected by a cutting line, it is split into disjoint rectangles whose union is the intersected rectangle. Thus, naturally, BSP is a *slice-and-dice* procedure since each "cut" separates a region into two subregions. The *size* of a BSP is the number of regions produced. Equivalently, a BSP of a set of rectangles contained in a rectangular region $\mathbf{R}$ can be visualized as a binary tree, where each node is a rectangular subregion of $\mathbf{R}$, each internal node is the union of its two children, the intersection of each leaf with the rectangles in our collection has at most one rectangle, and the root is the rectangular region $\mathbf{R}$. The partition of $\mathbf{R}$ that corresponds to a BSP is the collection of the leaves of this tree and the size of the BSP is the number of leaves in the tree. A set of rectangles form a tiling if they partition some rectangular region $\mathbf{R}$. It is shown in [17, 24] that a BSP of $\mathbf{R}$ with the minimum number of leaves can be computed using dynamic programming technique in $O(n^5)$ time.

Our improved results for the sizes of BSPs for isothetic rectangles are summarized in the following two theorems below. The amortization schemes in the proofs of these two theorems are somewhat different.

THEOREM 4.1. *We can compute a BSP of $\mathbf{R}$ containing at most $3n - \lceil \frac{3b}{4} \rceil$ regions in $O(n \log n)$ time, where $b \geq 2$ is the total number of rectangles adjacent to (having one side common) one of the four sides of $\mathbf{R}$.*

THEOREM 4.2. *Assume that the rectangles in our collection form a partition of $\mathbf{R}$. Then, we can compute a BSP of $\mathbf{R}$ containing at most $2n - \lceil \frac{3b}{4} \rceil$ regions in $O(n \log n)$ time, where $b \geq 3$ is the total number of rectangles adjacent to (having one side common) one of the four sides of $\mathbf{R}$.*

4.3 Approximate Tiling via BSP based Slice-and-Dice We show how the bounds in Theorems 4.1 and 4.2 can be used to provide improved approximation algorithms for general MIN-MAX and MAX-MIN tiling problems as defined in Section 4.1. Our new improved approximation results are as follows. The previously best known algorithm for this problem also used no more than twice as many tiles as needed, but its running time was $O(N^{\geq 5}\alpha)$ [17].

THEOREM 4.3. *Let p^* is the minimum number of tiles in an optimal solution of the general MIN-MAX problem. Then, there is an $O(N^{\frac{5}{2}}\alpha)$ time algorithm to find a tiling that uses at most $2p^* - 2$ tiles.*

The lower bound of $2n - o(n)$ on the size of BSPs forming a partition of their bounding box [11] indicates that alternate approach is needed in order to get significantly better approximations for this problem.

Our BSP techniques can also be applied to the general version of the MAX-MIN problem to obtain the following bicriteria approximation. Define a (γ, β)-approximation of the MAX-MIN tiling problem to be one which produces a tiling of $\mathbf{A}$ such that there are at least γp^* tiles of weight at least βW or more, where p^* is the maximum number of tiles used in an optimum solution. In this terminology, our result from Section 3 (Theorem 3.1) is an $(\frac{1}{3}, 1)$-approximation in general and $(\frac{2}{5}, 1)$-approximation for the case when the given array $\mathbf{A}$ is binary (for the specific weight function considered there).

THEOREM 4.4. *There exists a polynomial time (γ, β)-approximation algorithm for the MAX-MIN tiling problem for the following values of γ and β: (a) $\gamma = \frac{1}{2}$, $\beta = \frac{1}{4}$, (b) $\gamma = \frac{4}{9}$, $\beta = \frac{1}{3}$ and (c) $\gamma = \frac{1}{3}$, $\beta = \frac{1}{2}$.*

5 Shifted Slice and Dice with Dynamic Programming

In this section, we consider the general MIN-MAX tiling problem when the global aspect ratio of the rectangles in *some* optimum solution is bounded by b for some constant $b \geq 1$. The MIN-MAX problem is known to be NP-hard even when the global aspect ratio of the set of tiles in an optimal solution is at most 3 [24]. We use a slice-and-dice technique in which generation of the slices is inspired by the shifting technique used in [14, 15]; we call such an approach as a *shifted slice-and-dice* technique. Our results are summarized in the

theorem stated below. Notice that part (a) of the following theorem provides a $2 + o(1)$ approximation to the MIN-MAX tiling problem in *almost linear time* for this case.

THEOREM 5.1. *Assume that the global aspect ratio of the rectangles in some optimum solution is bounded by b for some constant $b \geq 1$. Then, the following results hold for the general* MIN-MAX *tiling problem:*

(a) *There exists an algorithm that runs in $O(\alpha N/\delta^5)$ time and returns a solution in which the total number of tiles used is at most $(2+\delta)$ times that of the optimum, for any $\delta > 0$.*
(b) *There exists a PTAS.*

Proof. We use the shifted slice-and-dice technique for both parts. First, we describe how to generate the slices. For any subarray $\mathbf{B}$ of $\mathbf{A}$, let OPT_B denote the minimum number of tiles needed for the general MIN-MAX problem on $\mathbf{B}$. Let X be the set of rectangles in an optimal solution such that the global aspect ratio of the rectangles in X is at most b. For notational simplicity, assume that b is an integer and the smallest side of any rectangle in X is 1; hence the largest side of any rectangle in X is at most b. Since X forms a partition of an $n \times n$ array A, all coordinates of all corner points of rectangles in X are from the set $\{1, 2, \ldots, n\}$.

Let ε be the constant involved in the PTAS, i. e., we are looking for a solution in which we use at most $(1+\varepsilon)$ times the number of tiles in an optimum solution. Let $\beta = \lceil \frac{1}{2\varepsilon} \rceil$. Consider the following β families $L_1, L_2, \ldots, L_\beta$ of vertical lines where L_j is the set of vertical lines $\{y = 1 + (j-1)b + cb\beta \,|\, c \in \mathbb{N}\}$. The following observations are true:

(a) Since the separation of any two lines in L_j is at least $b\beta$, no rectangle in X can cross two such lines.
(b) For any two vertical lines $\ell \in L_j$ and $\ell' \in L_k$ with $j \neq k$, the separation between them is at least b and hence no rectangle in X can cross both ℓ and ℓ'.

Let n_j be the number of rectangles in X crossing some vertical line in L_j. Observations (a) and (b) imply that $\sum_{j=1}^{\beta} n_j \leq OPT_A$. Hence, there exists a L_k such that $n_k \leq \frac{1}{\beta} OPT_A$. Our slices consist of the partition of $\mathbf{A}$ formed by the vertical lines in L_k. Assuming that for each slice we can solve the general MAX-MIN problem *exactly*, the union of these solutions uses at most $OPT_A + n_k \leq \left(1 + \frac{1}{\beta}\right) OPT_A$ rectangles.

Now, we can take take each of the above slices of size at most $n \times b\beta$ and apply the same technique in the horizontal directions by considering β families of horizontal lines such that at most $\frac{1}{\beta} OPT_A$ rectangles of X cross at least one such family. Hence, after this, we have β^2 families of slices of $\mathbf{A}$ each of size at most $b\beta \times b\beta$, such that if we can compute the solution to the general MAX-MIN problem for each slice in each family exactly then the union of the best solution over these families of slices will use at most $\left(1 + \frac{2}{\beta}\right) OPT_A \leq (1+\varepsilon) OPT_A$ rectangles.

Notice that there are $O(N/b^2\beta^2)$ slices in each family. Since each slice is of size $b\beta \times b\beta$, there are at most $O(b^4\beta^4)$ rectangles to consider for an exact solution and any exact solution of the slice can use no more than $b^2\beta^2$ rectangles. Hence, an exhaustive search for the best solution for each slice takes $O((b^4\beta^4)^{b^2\beta^2} \alpha b^2 \beta^2)$ time. Hence, the total time taken by our algorithm is at most $O((b^4\beta^4)^{b^2\beta^2} \alpha N \beta^2)$, which is $O(\alpha N)$ if b and β are constants. This proves part (b) of the theorem.

For part (a) of the theorem, we use the same technique as before, but we only approximately compute the solution within each slice of size $b\beta \times b\beta$ using the sparse and rounded dynamic programming techniques in [24] to reduce the running time at the expense of increasing the number of tiles. As a result, we use at most $(2+2\varepsilon) OPT_A$ rectangles, which is $(2+\delta) OPT_A$ by choosing $\delta = 2\varepsilon$. The time taken for each subarray is $O(b^5\beta^5\alpha)$ and hence the total time taken is at most $O(b^5\beta^5\alpha \frac{N}{b^2\beta^2}\beta^2) = O(\alpha N b^3 \beta^5) = O(\alpha N/\delta^5)$.

6 Applications of Tiling Results

As mentioned before, the tiling problems arise in numerous application areas. See [17, 24] for more details on these applications and overview of several others. Rather than repeating those application scenarios, here we list a set of applications that show variations of tiling problems that arise. We only provide sketchy details of the applications in this extended abstract due to lack of space.

Two Dimensional Histograms in Databases. Databases use histograms to approximate data distribution; this is used to estimate size of intermediate results under different query plans, and thereby choose efficient query execution plans. All commercial databases rely on histograms for this purpose, and this is a well-studied topic in database research (e.g., see [31]). We consider the case of two dimensional histograms where the input is a two dimensional array $\mathbf{A}$ representing two numerical attributes in the database, say, salary and age of employees. The entry $\mathbf{A}[i, j]$ is the number of employees in the database who have age i and salary j. The histogram is a tiling of $\mathbf{A}$. Two specific histograms of interest are the *equi-sum* [23] and *V-Opt* [31] histograms. In an equi-sum histogram, the weight of a tile is the

sum of all items in it and in a V-OPT, it is the sum of the squared difference between each item in the tile and the average of the items in the tile (denoted SSD). The goal in both is to minimize the number of tiles given a bound on either the maximum weight (in equi-sum histogram) or total weight (in V-OPT) of the tiles; this bound represents the error the database designer is willing to tolerate in estimation of the sizes of the result. It is easy to see that our techniques for solving the `MIN-MAX` problem apply directly for equi-sum histograms or can be shown to apply simply for V-OPT histograms. As a result, *two dimensional equi-sum and V-OPT histograms can be approximated in* $O(N^{\frac{5}{2}})$ *time within a factor of* 2. This substantially improves the previously best known results for this problem [24].

Two Dimensional Deviants. Suppose that we are given a two dimensional array **A**. The goal is again to tile **A** as in two dimensional histogram construction. However, a crucial difference is that *one is allowed to remove a subtile* (a subtile is an isothetic rectangle contained in a tile). Each such subtile is called a *deviant*. The weight of a deviant is the sum of the squared difference between each item in the tile and the average of the items in the tile (denoted SSD). The weight of a tile that is not a deviant is now computed as in SSD except that now it is the sum of squared differences between each of the remaining tile elements and their average spread over the total number of elements in the tile (including the deviants, if any, removed from the tile). Formally, the problem is as follows: given d, a bound on the number of deviants and the total weight W of tiles, find a tiling of **A** with a minimum number of tiles.[9] Using our techniques for the `MIN-MAX` tiling problem, we can show the following.

THEOREM 6.1. *There exists an* $O(N^{\frac{5}{2}}d^2)$ *time algorithm to solve the two dimensional deviant mining problem that returns a solution using at most* $4d$ *deviants and at most twice the optimum number of tiles.*

This is the first known efficient algorithm for the multidimensional deviants problem, and we are currently performing experiments on this problem using our result. Similar results can be obtained for another variation of this problem: constructing two dimensional histograms with "holes" called STHoles [6], but we omit the details here.

Covering Rectilinear Polygons with Holes and Don't Care Regions. Formally, this problem, which we abbreviate as the `RECT-DECOMP` problem, is as follows. We are given a rectilinear polygon P of n vertices (possibly with holes). The interior of P may contain a set of special disjoint rectilinear regions, which we term as the *don't care regions*. Our goal is to *partition* the interior of P, excluding the don't care regions, into a minimum number of rectangles. A rectangle in the partition must be included inside the given polygon, must not contain any holes and may or may not overlap any don't care region. The `RECT-DECOMP` problem without any don't care region can be solved in $O(n^{\frac{5}{2}})$ time *provided* no degenerate (point) holes are allowed [26, 27]. However, if degenerate holes are allowed, then the problem was shown to be NP-complete [18]. The `RECT-DECOMP` problem and its variations has applications in storing graphical images and in the manufacture of integrated circuits [14, 15, 21] and is also a natural combinatorial problem of independent interest. We present the first known approximation algorithm for this problem in its full generality, ie., including holes as well as don't care regions. The proof of the following theorem is similar to other results in Section 4 and is omitted from this extended abstract.

THEOREM 6.2. *Let* k *is the minimum number of rectangles in an optimal solution of the* `RECT-DECOMP` *problem for a rectilinear polygon* P *with* n *vertices. Then, there is an* $O(n^5)$ *time algorithm to find a solution to the* `RECT-DECOMP` *problem that uses at most* $3k-2$ *rectangles.*

7 Concluding Remarks

We have studied two general rectangular tiling problems, namely the `MAX-MIN` and the `MIN-MAX` problem. Using the slice-and-dice approach in several ways such as in a greedy manner, with Binary Space Partitions or with dynamic programming atop, we have obtained small constant factor approximations for these problems. For the `MAX-MIN` versions, our results are the first known; for the `MIN-MAX` versions, our results improve the running time and/or approximations of best known previous results. Even though a connoisseur may not truly appreciate the concept of cutting a *higher-dimensional Sicilian pie*, it would be of significant interest to study efficient approximation algorithms for these problems in higher dimensions. References [5, 33, 34] report some approximation results for the `MIN-MAX` prob-

[9] This is a technical definition that follows from [16]. The motivating observation there was that sometimes small islands of excessively large (or small) values adversely affect the approximation of a region. Therefore, it would be beneficial to consider them separately. The authors formalized that notion in terms of the deviants above, and showed that finding deviants not only led to better approximations but also served as an indication of certain interesting trends in the data and therefore was of mining value. They considered the one dimensional version of the problem and left two and higher dimensional problems open. Here, we have formalized the problem in two dimensions and obtained approximation results.

lem in higher dimensions, but unfortunately the approximation quality deteriorates linearly with an increase in the dimension.

References

[1] P. Agarwal, E. Grove, T. Murali and J. Vitter. Binary space partitions for fat rectangles. *Proc. Symp. Found. of Comp. Sci. (FOCS)*, 1996.

[2] P. Agarwal and S. Suri. Surface approximation and geometric partitions. *Proc. 5th ACM-SIAM Discrete Algorithms*, 24-33, 1994.

[3] C. Ballieux. Motion planning using binary space partitions. *Technical Report*, Utrecht University, 1993.

[4] P. Berman, B. DasGupta and S. Muthukrishnan. On the Exact Size of the Binary Space Partitioning of Sets of Isothetic Rectangles with Applications. Technical Report # 2000-26, DIMACS Center, Rutgers University, 2000.

[5] P. Berman, B. DasGupta, S. Muthukrishnan and S. Ramaswami. Improved Approximation Algorithms for Rectangle Tiling and Packing. 12th ACM-SIAM Symposium on Discrete Algorithms (SODA), January 2001, pp. 427-436.

[6] N. Bruno, S. Chaudhuri and L. Gravano. STHoles: A Multidimensional Workload-Aware Histogram. Proc. SIG on Management of Data (SIGMOD), 2001.

[7] S. Chin and S. Feiner. Near real-time shadow generation using BSP trees. *Computer graphics*, 23:99-106, 1989.

[8] F. d'Amore and P. G. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. *Information Proc. Letters*, 44, 255–259, 1992.

[9] M. de Berg. Linear size binary space partitions for fat objects. *Proc. 3rd Annual European Symp. on Algorithms*, LNCS 979, 252–263, 1995.

[10] M. de Berg, M. de Groot and M. Overmars. New results on binary space partitions in the plane. *Proc. 4th Scand. Workshop Algo. Theory*, Vol 824, LNCS, 61-72, 1994.

[11] A. Dumitrescu, J. Mitchell and M. Sharir. Binary Space Partitions for Axis-Parallel Segments, Rectangles and Hyperrectangles, to appear in 17th Annual ACM Symposium on Computational Geometry, June 2001.

[12] H. Fuchs, Z. Kedem and B. Naylor. On visible surface generation by a priori tree structures. *Comput. Graph.*, 14(3), 124-133, 1980.

[13] V. Hai Nguyen and P. Widmayer. Binary space partitions for sets of hyperrectangles. LNCS, 1023, 1995.

[14] D. Hochbaum. Approximation algorithms for NP-hard problems. PWS Publishing Company, Chapter 9, 1997.

[15] D. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of ACM*, 130-136, 1985.

[16] H. Jagadish, N. Koudas and S. Muthukrishnan. Mining Deviants in a Time Series Database. *Proc. Very Large Databases* (VLDB), 1999, 102-113.

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

[18] A. Lingas. The power of non-rectilinear holes. *Prof. 9th Intl. Colloq. Autom. Lang. Prog.,* (ICALP), 369-383, 1982.

[19] 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.

[20] F. Manne. Load Balancing in Parallel Sparse Matrix Computation. *Ph.D. Thesis*, Department of Informatics, University of Bergen, Norway, 1993.

[21] C. Mead and L. Conway. Introduction to VLSI Systems. Addison-Wesley, Reading, MA., 1980.

[22] J. Mitchell. On maximum flows in polyhedral domains, *J. Comput. Syst. Sci.*, 40, 88-123, 1990.

[23] M. Muralikrishna and D. DeWitt. Equi-depth histograms for estimating selectivity factors for multidimensional queries. *Proc. ACM SIGMOD*, 1988, 28–36.

[24] S. Muthukrishnan, V. Poosala and T. Suel. Rectangular Partitionings: Algorithms, Complexity and Applications. *Proc. Intl. Conf. on Database Theory*, 1999.

[25] B. Naylor and W. Thibault. Application of BSP trees to ray-tracing and CSG evaluation. Technical Report GIT-ICS 86/03, Georgia Tech., Feb 1986.

[26] T. Ohtsuki. Minimum Dissection of Rectilinear Polygons. *Proc. IEEE Symp. on Circuits and Systems* (1982), 1210-1213.

[27] L. Pagli, E. Lodi, F. Luccio, C. Mugnai and W. Lipski. On Two Dimensional Data Organization 2. *Fund. Inform.*, 2 (1979), 211-226.

[28] M. Paterson and F. Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete Comput. Geom.*, 5, 485-503, 1990.

[29] M. Paterson and F. Yao. Optimal binary space partitions for orthogonal objects. *J. Algorithms*, 13, 99-113, 1992.

[30] C. H. Papadimtriou, personal communication.

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

[32] H. Samet. *Applications of spatial data structures: computer graphics, image processing and GIS*, Addison-Wesley, 1990.

[33] J. Sharp. Tiling Multi-dimensional Arrays. *Foundations of Computing Theory* (FCT), 1999, 500-511.

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

[35] P. van Oosterom. A modified binary space partition for geographic information systems. *Int. J. GIS*, 4(2), 133-146, 1990.