Approximation Algorithms For MAX-MIN Tiling*

Piotr Berman† Bhaskar DasGupta‡ S. Muthukrishnan§

Abstract

The MAX-MIN tiling problem is as follows. We are given a two-dimensional array $\mathbf{A}[1,\ldots,n][1,\ldots,n]$ where each entry $\mathbf{A}[i][j]$ stores a *non-negative* number. Define a *tile* of $\mathbf{A}$ to be a subarray $\mathbf{A}[\ell,\ldots,r][t,\ldots,b]$ of $\mathbf{A}$, the *weight* of a tile to be the *sum* of all array entries in it, and a *tiling* of $\mathbf{A}$ to be a collection of tiles of $\mathbf{A}$ such that each entry $\mathbf{A}[i][j]$ is contained in *exactly* one tile.

Given a weight bound W the goal of our MAX-MIN tiling problem is to find a tiling of $\mathbf{A}$ such that

1. each tile is of weight *at least* W (the MIN condition), and
2. the number of tiles is *maximized* (the MAX condition).

The MAX-MIN tiling problem is known to be NP-hard; here, we present first non-trivial approximations algorithms for solving it.

*The preliminary version of these results appeared in 13^{th} ACM-SIAM Annual Symposium on Discrete Algorithms, January 6-8, 2002, pp. 455-464.

†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 Grants CCR-0296041, CCR-0206795 and CCR-0208749, and a UIC startup fund.

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

1 Introduction

We investigate a tiling problem to optimize a certain MAX-MIN criteria. This problem is natural, well-motivated and is known to be NP-hard. Our main results are non-trivial approximation algorithms for this problem using variations of a simple slice and dice approach. In what follows, we present our problem and results formally.

1.1 Formal Problem Statement

Our input is a two dimensional array $\mathbf{A}[1,\ldots,n][1,\ldots,n]$ in which each entry $\mathbf{A}[i][j]$ stores a *non-negative* number[1] and a weight bound W. Define a *tile* of $\mathbf{A}$ to be a subarray $\mathbf{A}[\ell,\ldots,r][t,\ldots,b]$ of $\mathbf{A}$, the *weight* of a tile to be the *sum* of all array entries in it, and a *tiling* of $\mathbf{A}$ to be a collection of tiles of $\mathbf{A}$ such that each entry $\mathbf{A}[i][j]$ is contained in *exactly* one tile. The goal of our MAX-MIN tiling problem is to find a tiling such that

1. each tile is of weight *at least* W (the MIN condition), and
2. 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, hence we will assume this to be true. This MAX-MIN tiling problem is quite a natural optimization problem. Typically, they arise when one wants to only consider certain 'classes" that are each sufficiently significant, an wishes to minimize the "classes" under consideration. Such situations arise for example in data mining: mining optimized association rules with numerical attributes [4]. This problem was posed to us in a personal communication [7].

There is a related tiling problem, namely, the MIN-MAX tiling problem of [2, 3] where one is provided a budget of number of tiles, and the goal is to maximize the weight of tile in a tiling subject to the budget. Both MIN-MAX and MAX-MIN problems are known to be NP-hard [5]. Although our MAX-MIN tiling problem may appear to be similar to the MIN-MAX tiling problem, there are fundamental differences. For example, if we take a feasible solution for the MIN-MAX tiling problem and further divide some of the tiles, the solution still remains a feasible solution; hence a tile that is different from the optimal one may be "fixed" with a few additional tiles as is typically done in approximation algorithms for the MIN-MAX problem. In contrast, this property does not hold for our MAX-MIN tiling problem, and as a result it is crucial that the slices be almost perfect[2].

1.2 Our Results

Define a (r, s)-approximation of the MAX-MIN tiling problem to be a solution that produces a tiling of $\mathbf{A}$ with *at least* rp^* tiles each of which has weight *at least* $s(W - c)$, where p^* is the

[1] Unless otherwise stated, we will use **bold** letters to denote arrays/rectangles and respective regular (non-bold) letters to denote their weights. In particular, input array $\mathbf{A}$ has weight A, and $\mathbf{R}_i$, the i^{th} row of $\mathbf{A}$, has weight R_i.

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

maximum number of tiles used in an optimum solution and $c \geq 0$ is a constant. The input 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. Let m be the number of non-zero elements of $\mathbf{A}$. The following table summarizes our approximation results.

restriction (if any)	(r, s)-approximation	time	Theorem
$\mathbf{A}$ is arbitrary	$(\frac{1}{3}, 1)$	$O(m+n)$	Theorem 1
	$(\frac{4}{9}, \frac{1}{3})$	$O(n^7)$	Theorem 3
	$(\frac{1}{2}, \frac{1}{4})$	$O(n^7)$	Theorem 3
$\mathbf{A}$ is binary	$(\frac{2}{5}, 1)$	$O(m+n)$	Theorem 1

Figure 1: A summary of the results in this paper.

1.3 Techniques

We use variations of what we informally refer to as "slice-and-dice" technique (or simply slicing and dicing). Researchers have previously used variations or specific implementations of this slice-and-dice techniques to obtain approximation algorithms for other tiling problems (for example, see [2, 3, 5, 8, 9]). While the slice-and-dice technique by itself is conceptually and algorithmically simple, the crux of our technical work is in the analyses. In general, an *informal* description of this 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.
- 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.

We use the following two versions of the slice-and-dice approach:

(Greedy Slice-and-Dice) The algorithm for Theorem 1 is based on *greedily* slicing the array into strips and dicing each slice; the resulting running time is *linear* in the number of non-zero elements of $\mathbf{A}$ and is very simple to implement. Our greedy slice-and-dice algorithm is

similar in flavor to that for the MIN-MAX tiling problem of [2, 3]; however, details, analysis and lower bounds used are all quite different.

(Recursive Slice-and-Dice: Binary Space Partitionings) We use this approach for the algorithm in Theorem 3. A *recursive* application of the slice-and-dice technique to the MAX-MIN tiling problem can be done via a *binary space partitioning* (BSP) of the tiles. Therefore, this approach to solving the MAX-MIN tiling problem uses a BSP of isothetic rectangles where the size of the BSP affects the quality of approximation. For MAX-MIN tiling 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.

1.4 Map

We present our approximation algorithms using the greedy slice-and-dice technique in Section 2. In Section 3 we provide approximation algorithms using a recursive slice-and-dice technique. Finally, in Section 4, we present our concluding remarks with possible future research directions.

2 Approximating MAX-MIN Tiling Via Greedy Slice and Dice

If the input array $\mathbf{A}$ contains m non-zero entries 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[3]. For this section, we assume that our input array is represented this way.

We may assume that the weight bound W is 1 by scaling all the entries of $\mathbf{A}$, if necessary. Now, a *feasible* solution for the MAX-MIN tiling problem is a tiling of $\mathbf{A}$ in which every tile has a weight of at least 1. Obviously, we may assume that $A \geq 1$, since otherwise the MAX-MIN tiling problem has no feasible solution[4]. Let $b = \max_{1 \leq i,j \leq n} \mathbf{A}[i][j]$. We may assume that $b \leq 1$ by using the following crucial observation.

Observation 1 *Given an instance array* $\mathbf{A}$ *of the* MAX-MIN *tiling 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 the following theorem giving a $(\frac{1}{3}, 1)$-approximation to the MAX-MIN tiling problem.

[3] If the m non-zero entries of the array $\mathbf{A}$ are given in some arbitrary order, then they can be represented as an array of row lists using bucket sort in $O(m + n)$ time. This will not increase the overall time complexity of our algorithms.

[4] Remember that A is the weight (sum of all elements) of the input array $\mathbf{A}$ by our notation.

Theorem 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 each of weight at least* W*, where* p^* *is the (maximum) number of tiles used by an optimal algorithm and* $r = \begin{cases} \frac{5}{2} & \text{if } \mathbf{A} \text{ is binary} \\ 3 & \text{otherwise} \end{cases}$

Proof. First, we describe a basic *slicing* algorithm that is used by the algorithm in this proof. 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*, all other non-remainder slices are called *regular* slices. More precisely, the algorithm computes ℓ and array entries $t[0] = 0, t[1], \ldots, t[\ell]$, so that each regular slice $\mathbf{S}_i$ (for $1 \leq i \leq \ell$) consists of rows $\mathbf{R}_{t[i-1]+1}$ to $\mathbf{R}_{t[i]}$, while the remainder slice $\mathbf{S}_{\ell+1}$ starts at row $\mathbf{R}_{t[\ell]+1}$. A psuedocode of the slicing algorithm is as shown below; it is easy to see that this algorithm can be implemented in $O(n+m)$ time.

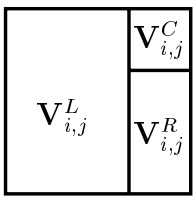
```
t[0] ← 0
ℓ ← 0
slice_weight ← 0
for i ← 1 to n do
    slice_weight ← slice_weight + R_i
    if slice_weight ≥ 1 then
        ℓ ← ℓ + 1
        t[ℓ] ← i
        slice_weight ← 0
```

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 the slicing algorithms as described above. Our dicing step will then consist of partitioning the union of regular 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 $\alpha(i)$, so the V-slices $\mathbf{V}_{i,1}, \ldots \mathbf{V}_{i,\alpha(i)}$ are regular and $\mathbf{V}_{i,\alpha(i)+1}$ is the *remainder* V-slice. To obtain our preliminary partition, we combine each remainder V-slice with its preceeding regular V-slice. Obviously, this part of dicing can also be done in a total of $O(n+m)$ time.

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



We can estimate the weights of these parts as follows:

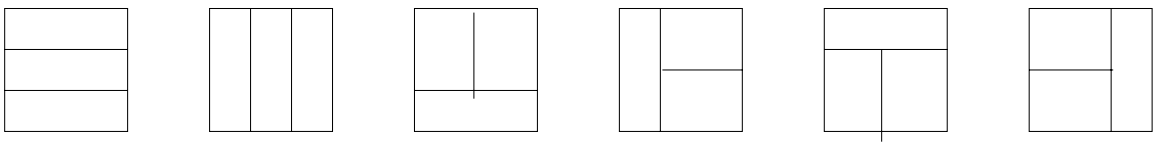
$V^L_{i,j} < 1$ because $\mathbf{V}^L_{i,j}$ did not make a complete V-slice,
$\sum_{j=1}^{\alpha(i)} V^R_{i,j} < 1$ because all $\mathbf{V}^R_{i,j}$'s together for a specific i did not make a complete slice,
$V^C_{i,j} \leq 1$ because no array entry exceeds 1.

Consequently, the total weight S_i of the slice $\mathbf{S}_i$ satisfies $S_i = V_{\alpha(i)+1} + \sum_{j=1}^{\alpha(i)}(V^L_{i,j} + V^R_{i,j} + V^C_{i,j}) < 2 + 2\alpha(i)$. Notice that these splits of the V-slices of each regular slice can also be done in a total of $O(n+m)$ time.

Before we continue, observe that we can already guarantee to use at least $\lfloor A \rfloor/4$ tiles. Note that $S_i < 2 + 2\alpha(i) \leq 4\alpha(i)$ and $A < 1 + \sum_{i=1}^{\ell} S_i \leq 1 + \sum_{i=1}^{\ell} 4\alpha(i)$, while we used $t = \sum_{i=1}^{\ell} \alpha(i)$ tiles. Therefore $4t > A - 1$ and hence $4t \geq \lfloor A \rfloor$, proving immediately an approximation ratio of 4. To improve this ratio, we do the following *additional* dicing:

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

To ensure a total running time of $O(n+m)$, we need to show how to implement each such additional dicing on $\mathbf{S}_{i-1} \cup \mathbf{S}_i$ (whenever needed) in $O(\beta + 2)$ time, where β is the total number of non-zero elements of $\mathbf{S}_{i-1} \cup \mathbf{S}_i$. Since $3 \leq \alpha(i-1) + \alpha(i) + 1 \leq 4$ when this additional dicing step is necessary, it suffices to show how to implement this step in $O(\beta + 2)$ time when $\mathbf{S}_{i-1} \cup \mathbf{S}_i$ is partitioned into 4 good rectangles (a partition into 3 good rectangles can be obtained in an intermediate step in the solution for a partition into 4 good rectangles). An inspection of Cases 6, 7, 8, 10, 11 and 12 show that the tiles made by the additional dicing step is only one of the following types: some slice in the current partition of $\mathbf{S}_{i-1}$ (respectively, $\mathbf{S}_i$) extended vertically to cover rows of $\mathbf{S}_i$ (respectively, $\mathbf{S}_{i-1}$), a rectangular part of a current slice of $\mathbf{S}_i$ or $\mathbf{S}_{i-1}$ and a tiling of a remaining rectangular region of $\mathbf{S}_{i-1} \cup \mathbf{S}_i$ into at most 3 rectangles. The first two types are easy to compute, given the current partitioning of $\mathbf{S}_{i-1}$ and $\mathbf{S}_i$, in $O(\beta + 2)$ time. Hence, it is sufficient to show how to partition a given rectangle into 3 good rectangles (if possible). There are only six basic ways to partition a given rectangle into 3 good rectangles as shown below and it is easy to verify if one of these partitions exist using a greedy approach in $O(\beta + 2)$ time.



To show that we will use t good tiles such that $t \geq \left\lfloor \frac{A+1}{3} \right\rfloor$, it suffices to show that $A < 3t + 2$. This in turn follows from $\sum_{i=1}^{\ell} S_i < 3t + 1$.

Define $\sigma(1) = 0$ and $\sigma(i)$, for $2 \leq i \leq \ell$ to be as follows:

$$\sigma(i) = \begin{cases} 1 & \text{if the above } \textbf{for} \text{ loop increased the number of tiles while processing } \mathbf{S}_{i-1} \cup \mathbf{S}_i \\ 0 & \text{otherwise} \end{cases}$$

Clearly, our algorithm uses $t = \sum_{i=1}^{\ell} [\alpha(i) + \sigma(i)]$ tiles, so we need to show that $\sum_{i=1}^{\ell} [S_i - 3\alpha(i) - 3\sigma(i)] < 1$. For notational convenience, let $\zeta(k) = \sum_{i=1}^{k} [S_i - 3\alpha(i) - 3\sigma(i)]$ for $1 \leq k \leq \ell$ and

$\zeta(0) = 0$. Hence, in our new notation, it is sufficient to show that $\zeta(\ell) < 1$. We do it by induction on ℓ. Our inductive claim is the following:

> if $\zeta(k) > 0$ then $\alpha(k) = 1$, $S_i - V^L_{k,1} > 1$ and $\zeta(k) < V^L_{k,1} + V_{k,2} - 1$

This inductive claim is sufficient, because for $k = \ell$, if $\zeta(k) > 0$ then

- $V^L_{k,1} < 1$ as has already been pointed out before,
- $V_{k,2} \leq 1$ because, by inductive claim, $\alpha(k) = 1$ and hence $\mathbf{V}_{k,2}$ is the remainder slice,
- and, hence, $\zeta(k) < 1$.

The basis of induction is trivial since $\zeta(0) = 0$. For the inductive step, we assume that the claim holds for $\zeta(k-1)$ and then we consider several cases.

Case 1: $\alpha(k) > 2$**.** Then, $\zeta(k) < \zeta(k-1) + S_k - 3\alpha(k) < 1 + 2 + 2\alpha(k) - 3\alpha(k) = 3 - \alpha(k) \leq 0$.

Case 2: $\zeta(k-1) \leq 0$ **and** $\alpha(k) = 2$**.** Then, $\zeta(k) < \zeta(k-1) + S_k - 3\alpha(k) < S_k - 3\alpha(k) < 2 + 4 - 6 = 0$.

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

Case 4: $\sigma(k) = 1$**.** Then, $\zeta(k) < \zeta(k-1) + S_k - 3\alpha(k) - 3 < 1 + (2 + 2\alpha(k)) - 3\alpha(k) - 3 = -\alpha(k) \leq -1$.

For the remaining cases, we can assume that $\zeta(k-1) > 0$, $\sigma(k) = 0$ and $\alpha(k) \in \{1, 2\}$. First we consider various cases for $\alpha(k) = 1$. For convenience, we will use the following notations:

> $\mathbf{C} = \mathbf{V}^L_{k-1,1}$, $\mathbf{D} = \mathbf{V}^C_{k-1,1} \cup \mathbf{V}^R_{k-1,1}$, $\mathbf{E} = \mathbf{V}_{k-1,2}$, $\mathbf{C}' = \mathbf{V}^L_{k,1}$, $\mathbf{D}' = \mathbf{V}^C_{k,1} \cup \mathbf{V}^R_{k,1}$, and $\mathbf{E}' = \mathbf{V}_{k,2}$

Since $\zeta(k-1) > 0$, by inductive hypothesis $\zeta(k-1) < C + E - 1$, $S_{k-1} - C > 1$ and $\alpha(k-1) = 1$.

Case 5: $S_k - C' < 1$**.** Then, $S_k < 2$ and $\zeta(k) < 1 + 2 - 3 = 0$.

Case 6: $S_k - C' \geq 1$ **and $\mathbf{D}$ and $\mathbf{D}'$ are in the same column of the given array.** One possible way to partition $\mathbf{S}_{k-1} \cup \mathbf{S}_k$ is using the three tiles[5] $\mathbf{C} \cup \mathbf{C}'$, $\mathbf{S}_{k-1} - \mathbf{C}$, and $\mathbf{S}_k - \mathbf{C}'$. The tile $\mathbf{S}_{k-1} - \mathbf{C}$ is good since $S_{k-1} - C > 1$. and the tile $\mathbf{S}_k - \mathbf{C}'$ is good since $S_k - C' \geq 1$. But, since $\sigma(k) = 0$, one of the three tiles must not be good, therefore we must have $C + C' < 1$. By a symmetric argument, $E + E' < 1$. Thus $\zeta(k) < C + E - 1 + C' + D' + E' - 3 < C' - 2 < 0$.

For the remaining cases we use the additional notations: $\mathbf{D}'_u = \mathbf{V}^C_{k,1}$ and $\mathbf{D}'_d = \mathbf{V}^R_{k,1}$.

[5]For two tiles $\mathbf{X}$ and $\mathbf{Y}$ such that $\mathbf{Y}$ is a part of $\mathbf{X}$, $\mathbf{X} - \mathbf{Y}$ is the tile obtained by removing $\mathbf{Y}$ from $\mathbf{X}$.

Case 7: $\mathbf{D}'$ is in a column of $\mathbf{C}$. One attempt to partition $\mathbf{S}_{k-1} \cup \mathbf{S}_k$ is to use the following three tiles: extension of $\mathbf{C}$ upward to cover $\mathbf{D}'_d$, the top row of $\mathbf{S}_k$ and the remaining uncovered part of $\mathbf{S}_{k-1} \cup \mathbf{S}_k$. If the top row of $\mathbf{S}_k$ is not good, then that already implies $S_k < 2$ and hence $\zeta(k) < 1 + 2 - 3 = 0$. So, assume that the top row of $\mathbf{S}_k$ is good. Since $S_{k-1} - C > 1$, the remaining uncovered part of $\mathbf{S}_{k-1} \cup \mathbf{S}_k$, which includes $S_{k-1} - C > 1$, is also good. But, since $\sigma(k) = 0$, one of the three tiles must not be good, therefore the extension of $\mathbf{C}$ upward to cover $\mathbf{D}'_d$ cannot be good, implying $C + D'_d < 1$, and thus $\zeta(k) < (C + E - 1) + (C' + D'_d + D'_u + E') - 3 < E + C' + D'_u + E' - 3 < C' + E' - 1$.

Case 8: $\mathbf{D}'$ is in a column of $\mathbf{E}$. Symmetric to Case 7.

For the remaining cases we assume that $\alpha(k) = 2$, so we have to show $\zeta(k) < 0$. We use all the notations as before, except that now $\mathbf{F}'_d = \mathbf{V}^R_{k,2}$.

Case 9: $S_k < 5$**.** This is similar to Case 5 since now $\zeta(k) < \zeta(k-1) + S_k - 3\alpha(k) < 1 + 5 - 6 = 0$.

Case 10: $S_k \geq 5$ **and $\mathbf{D}$ and $\mathbf{D}'$ are in the same column.** One attempt to partition $\mathbf{S}_{k-1} \cup \mathbf{S}_k$ into at least $\alpha(k-1) + \alpha(k) + 1 = 4$ tiles is $\mathbf{C} \cup \mathbf{C}'$, $\mathbf{S}_{k-1} - \mathbf{C}$, and two tiles made of $\mathbf{S}_k - C'$. If we cannot partition $\mathbf{S}_k - \mathbf{C}'$ into two good tiles, then $S_k - C' < 4$ and thus $S_k < 5$, an impossibility for this case. The tile $\mathbf{S}_{k-1} - \mathbf{C}$ is also good since, by inductive hypothesis, $S_k - C' > 1$. But, this attempt must fail since $\sigma(k) = 0$. Hence, 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 11: $S_k \geq 5$ **and $\mathbf{D}$ and $\mathbf{F}'_d$ are in the same column.** Symmetric to Case 10.

Case 12: when none of the above 11 **cases apply.** Now neither $\mathbf{D}'$ nor $\mathbf{F}'_d$ is in the same column as $\mathbf{D}$. To make analysis more succinct, we introduce a some more notations. First, let $X = S_k - D'_d - F'_d$. Then $X = C' + D'_u + V^L_{k,2} + V^C_{k,2} + V_{k,3} < 5$. Second, Y (respectively, Z) is the sum of weights of those elements of $\mathbf{D}'_d$ and $\mathbf{F}'_d$ (if any) that are in the same column as $\mathbf{C}$ (respectively, $\mathbf{E}$). Obviously, $S_k = X + Y + Z$.

One attempt to partition $\mathbf{S}_{k-1} \cup \mathbf{S}_k$ using at least $\alpha(k-1) + \alpha(k) + 1 = 4$ tiles is one tile extending $\mathbf{C}$ upward towards $\mathbf{S}_k$ so that it covers all but the topmost row of $\mathbf{S}_k$, two tiles covering the top row of $\mathbf{S}_k$, and one tile covering the rest of $\mathbf{S}_{k-1} \cup \mathbf{S}_k$. Since $S_k \geq 5$, the total weight of the top row of $\mathbf{S}_k$ is at least 4 and hence it is possible to partition the top row of $\mathbf{S}_k$ into two good tiles. The tile covering the rest of $\mathbf{S}_{k-1} \cup \mathbf{S}_k$ includes $\mathbf{S}_{k-1} - \mathbf{C}$ and hence is good since $S_{k-1} - C > 1$ by inductive hypothesis. But, since $\sigma(k) = 0$, this attempt must fail. Therefore, the tile extending $\mathbf{C}$ upward towards $\mathbf{S}_k$ so that it covers all but the topmost row of $\mathbf{S}_k$ is not good, hence $C + Y < 1$. Similarly, we can show that $E + Z < 1$. Thus $\zeta(k) < \zeta(k-1) + S_k - 3\alpha(k) < C + E - 1 + S_k - 6 = (C + Y) + (E + Z) + X - 7 < 0$.

This ends the analysis of the algorithm for the case when $\mathbf{A}$ is arbitrary. Now we consider the case when $\mathbf{A}$ is binary. After rescaling the entries of $\mathbf{A}$ such that $W = 1$, we may assume that each non-zero element of $\mathbf{A}$ is equal to W^{-1} where W is an integer. We will prove that $A < \frac{5t+3}{2}$. The algorithm and the proof that $A < \frac{5t+3}{2}$ is similar to the previous, so we will only list the differences.

- $V^R_{i,j}, V_{i,\alpha(i)+1} \leq 1 - \frac{1}{W}$, $V^C_{i,j} \leq \frac{1}{W}$, $\sum_{j=1}^{\alpha(i)} V^R_{i,j} \leq 1 - \frac{1}{W}$, thus $S_i \leq \alpha(i) + 2 - \frac{2}{W}$.
- We will show that the number t of tiles produced by our algorithm satisfies $A < \frac{5t+3}{2}$ by showing that $\zeta(k) = \sum_{i=1}^{k}[S_i - \frac{5}{2}\alpha(i) - \frac{5}{2}\sigma(i)] < 1/2$.
- Our modified inductive claim is: if $\zeta(k) > 0$ then $\alpha(k) = 1$ and $\zeta(k) \leq S_k - \frac{5}{2}$. Note that this means that $\zeta(k) \leq \frac{1}{2} - \frac{2}{W}$.
- The basis of induction is trivial as before by defining $\zeta(0) = 0$. We now have the following simpler case analysis for the inductive step:

  Case 1: $\alpha(k) > 1$**.** Then, $\zeta(k) < \frac{1}{2} + S_k - \frac{5}{2} < \alpha(k) + 2 - \frac{5}{2}\alpha(k) < 2 - \alpha(k) \leq 0$.

  Case 2: $\alpha(k) = 1$ **and** $\zeta(k-1) \leq 0$. If $\zeta(k) > 0$ then $\zeta(k) \leq \zeta(k-1) + S_k - \frac{5}{2}\alpha(k) \leq S_k - \frac{5}{2}$.

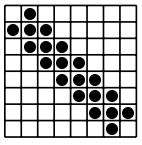
  Case 3: $\alpha(k) = 1$ **and** $\zeta(k-1) > 0$**.** By inductive hypothesis, $\alpha(k-1) = 1$ and $\zeta(k-1) \leq S_{k-1} - \frac{5}{2}$. If $\zeta(k) \leq S_k - \frac{5}{2}$, then there is nothing to prove. Otherwise, $\zeta(k) > S_k - \frac{5}{2}$. Since $\zeta(k) = \zeta(k-1) + S_k - \frac{5}{2} - \frac{5}{2}\sigma(k)$, it follows that

  $$S_{k-1}+S_k-4 = \left(S_{k-1} - \frac{5}{2}\right)+\left(S_k - \frac{3}{2}\right) \geq \zeta(k-1)+\left(S_k - \frac{3}{2}\right) = \zeta(k)+1+\frac{5}{2}\sigma(k) > 1$$

  implying $S_{k-1} + S_k > 5$. Because the total weight of elements of all the rows of a slice, excluding the elements of the top row, is at most $1 - \frac{1}{W}$, the sum of weights of the elements of the top rows of $\mathbf{S}_{k-1}$ and $\mathbf{S}_k$ is at least $3 + \frac{2}{W}$. We now 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-zeros from the top rows, and the previous columns have at most $W - 1$ non-zeros, thus the weight of the intersection of a vertical slice with the two top rows is at most $1 + \frac{1}{W}$. Thus after creating the first 2 vertical slices we still have a total weight of at least 1 available among remaining uncovered elements in the two top rows. Consequently, we are successful to partition $S_{k-1} \cup S_k$ into $\alpha(k-1) + \alpha(k) + 1 = 3$ tiles. Therefore, $\sigma(k) = 1$ and $\zeta(k) \leq S_{k-1} + S_k - \frac{15}{2} < 0$.

□

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 • 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.



3 Approximating MAX-MIN Tiling Via Recursive (BSP Based) Slice and Dice

First, we need to review the definitions and related results for binary space partitioning (BSP) of a set of isothetic rectangles. Then, we show how to use these results for our approximation algorithm.

3.1 Binary Space Partitioning: Definitions and Results

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. A set of rectangles form a tiling if they partition some rectangular region $\mathbf{R}$. It is shown in [5,6] that a BSP of $\mathbf{R}$ with the minimum number of regions can be computed using dynamic programming technique in $O(n^5)$ time.

Following is the improved result for the sizes of BSPs for isothetic rectangles as obtained in [1] that will be of use to us.

Theorem 2 [1] *Assume that the rectangles in our collection form a partition of* $\mathbf{R}$. *Then, there exists a BSP of* $\mathbf{R}$ *containing at most* $2n-1$ *regions. Moreover, in such a BSP each rectangle rectangle of* $\mathbf{R}$ *is cut into at most* 4 *pieces.*

3.2 Our Approximation Algorithm

The following theorem is our main result in this section.

Theorem 3 *There exists an* $O(n^7)$ *time* (r,s)*-approximation algorithm for the* MAX-MIN *tiling problem for the following values of* r *and* s*: (a)* $r=\frac{1}{2}$ *and* $s=\frac{1}{4}$ *and (b)* $r=\frac{4}{9}$ *and* $s=\frac{1}{3}$.

Proof. Consider an optimum solution OPT that uses p^* tiles each of weight at least W. By Theorem 2 there exists a BSP of OPT that has at most $2p^*$ tiles in which each rectangle of OPT is cut into at most 4 pieces. Let $w_i \geq 0$ be the *fraction* of rectangles in OPT that is cut into i pieces for $1 \leq i \leq 4$ (hence, $\sum_{i=1}^{4} w_i = 1$). Hence, $\sum_{i=1}^{4} iw_i \leq 2$. which implies $w_4 \leq \frac{1}{2}$. Assume that $w_4 = \frac{1}{2} - \varepsilon$ for some fraction $\frac{1}{2} \geq \varepsilon \geq 0$. Then, $\sum_{i=1}^{3} w_i = 1 - w_4 = \frac{1}{2} + \varepsilon$ and $\sum_{i=1}^{3} iw_i \leq 2 - 4w_4 = 4\varepsilon$. The last inequality implies that $w_3 \leq \frac{4}{3}\varepsilon$. Assume that $w_3 = \frac{4}{3}\varepsilon - \varepsilon_1$ for some fraction $\frac{4}{3}\varepsilon \geq \varepsilon_1 \geq 0$. Hence, $\sum_{i=1}^{2} w_i = \frac{1}{2} + \varepsilon - w_3 = \frac{1}{2} - \frac{\varepsilon}{3} + \varepsilon_1$ and $\sum_{i=1}^{2} iw_i \leq 4\varepsilon - 3w_3 = 3\varepsilon_1$. The last inequality implies that $w_2 \leq \frac{3}{2}\varepsilon_1$. Assume that $w_2 = \frac{3}{2}\varepsilon_1 - \varepsilon_2$ for some fraction $\frac{3}{2}\varepsilon_1 \geq \varepsilon_2 \geq 0$. Hence, $w_1 = \frac{1}{2} - \frac{\varepsilon}{3} + \varepsilon_1 - w_2 = \frac{1}{2} - \frac{\varepsilon}{3} - \frac{\varepsilon_1}{2} + \varepsilon_2$. If a rectangle of weight W is cut into i pieces, then obviously at least one of the piece has a weight of at least $\frac{W}{i}$. Hence,

the fraction k of rectangles of OPT which has a weight of at least $\frac{W}{4}$ in the BSP is at least

$$\begin{array}{rcl} k & \geq & \sum_{i=1}^{4} \frac{w_i}{i} \\ & = & \left(\frac{1}{2} - \frac{\varepsilon}{3} - \frac{\varepsilon_1}{2} + \varepsilon_2\right) + \frac{1}{2}\left(\frac{3}{2}\varepsilon_1 - \varepsilon_2\right) + \frac{1}{3}\left(\frac{4}{3}\varepsilon - \varepsilon_1\right) + \frac{1}{4}\left(\frac{1}{2} - \varepsilon\right) \\ & = & \frac{5}{8} - \frac{5\varepsilon}{36} - \frac{\varepsilon_1}{12} + \frac{\varepsilon_2}{2} \geq \frac{5}{8} - \frac{5\varepsilon}{36} - \frac{\varepsilon_1}{12} \geq \frac{5}{8} - \frac{5}{72} - \frac{1}{18} = \frac{1}{2} \quad (\text{since } \varepsilon \leq \frac{1}{2} \text{ and } \varepsilon_1 \leq \frac{4\varepsilon}{3} \leq \frac{2}{3}) \end{array}$$

This provides a $(\frac{1}{2}, \frac{1}{4})$-approximation. Similarly, the fraction k' of rectangles of OPT which has a weight of at least $\frac{W}{3}$ in the BSP is at least

$$\begin{array}{rcl} k' & \geq & \sum_{i=1}^{3} \frac{w_i}{i} \\ & = & \left(\frac{1}{2} - \frac{\varepsilon}{3} - \frac{\varepsilon_1}{2} + \varepsilon_2\right) + \frac{1}{2}\left(\frac{3}{2}\varepsilon_1 - \varepsilon_2\right) + \frac{1}{3}\left(\frac{4}{3}\varepsilon - \varepsilon_1\right) \\ & = & \frac{1}{2} + \frac{\varepsilon}{9} - \frac{\varepsilon_1}{12} + \frac{\varepsilon_2}{2} \geq \frac{1}{2} - \frac{\varepsilon_1}{12} \geq \frac{1}{2} - \frac{1}{18} = \frac{4}{9} \end{array}$$

giving a $(\frac{4}{9}, \frac{1}{3})$-approximation. To compute such a BSP of $\mathbf{A}$, we use a natural dynamic programming technique similar to that used in [5] for a specific $\frac{1}{s} \in \{3, 4\}$. Define a hierarchical binary partition (HBP) of an array to be either the entire array or the unions of the HBPs of the two subarrays of a partition of the array by a horizontal or vertical cut. Obviously, a BSP of OPT is also a HBP of the given array $\mathbf{A}$. For each subarray $\mathbf{A}[i, \ldots, j][k, \ldots, \ell]$ and for each $1 \leq p \leq n^2$, define $\Gamma(i, j, k, \ell, p)$ to be a HBP of this subarray of in which p tiles are of weight at least sW if such a partition exists, and to be $\emptyset$ otherwise. There are at most $O(n^4)$ subarrays of the given array $\mathbf{A}$, and it is easy to calculate the weights of all of these subarrays using dynamic programming in $O(n^4)$ time. If the weight of the subarray $\mathbf{A}[i, \ldots, j][k, \ldots, \ell]$ is less than sW, then obviously $\Gamma(i, j, k, \ell, p) = \emptyset$ for every p. On the other hand, if the weight of the subarray $\mathbf{A}[i, \ldots, j][k, \ldots, \ell]$ is at least sW, then there are $(j - i) + (\ell - k) + 3 \leq 2n - 1$ ways to construct a HBP of $\mathbf{A}[i, \ldots, j][k, \ldots, \ell]$:

- The entire subarray as a single tile. This implies that $\Gamma(i, j, k, \ell, 1) = \mathbf{A}[i, \ldots, j][k, \ldots, \ell]$.
- Split the subarray by a vertical line into two subarrays $\mathbf{A}[i, \ldots, j][k, \ldots, \ell_1]$ and $\mathbf{A}[i, \ldots, j][\ell_1 + 1, \ldots, \ell]$, or by a horizontal line into two subarrays $\mathbf{A}[i, \ldots, j_1][k, \ldots, \ell]$ and $\mathbf{A}[j_1+1, \ldots, j][k, \ldots, \ell]$. By dynamic programming, we we know the values of $\Gamma(i, j, k, \ell_1, p)$, $\Gamma(i, j, \ell_1 + 1, \ell, p)$, $\Gamma(i, j_1, k, \ell, p)$ and $\Gamma(j_1 + 1, j, k, \ell, p)$ for all p, hence we can use them to find the value of $\Gamma(i, j, k, \ell, p)$ for each p.

The final answer is contained in $\{\Gamma(1, n, 1, n, p) \mid 1 \leq p \leq n^2\}$. The total time taken by our algorithm is $O(n^4 \cdot n \cdot n^2) = O(n^7)$. □

4 Concluding Remarks

We studied the `MAX-MIN` tiling problem and obtained non-trivial approximation algorithms for this problem using the slice-and-dice approach in two ways. It would be of significant interest to study efficient approximation algorithms for this problem in higher dimensions and we expect variations of the slice-and-dice technique to be useful there as well.

References

[1] P. Berman, B. DasGupta and S. Muthukrishnan. On the Exact Size of the Binary Space Partitioning of Sets of Isothetic Rectangles with Applications. *SIAM Journal of Discrete Mathematics*, Vol. 15, No. 2, pp. 252-267, 2002.

[2] P. Berman, B. DasGupta, S. Muthukrishnan and S. Ramaswami. Improved Approximation Algorithms for Rectangle Tiling and Packing. 12^{th} *ACM-SIAM Symposium on Discrete Algorithms*, pp. 427-436, 2001.

[3] P. Berman, B. DasGupta, S. Muthukrishnan and S. Ramaswami. Efficient Approximation Algorithms for Tiling and Packing Problems With Rectangles. *Journal of Algorithms*, Vol. 41, pp. 443-470, 2001.

[4] T. Fukuda, Y. Morimoto, S¿ Morishita and T. Tokuyama. Mining optimized association rules for numeric attributes. *ACM Proc. on Principles of Database Systems (PODS)*, pp 182-191, 1996.

[5] S. Khanna, S. Muthukrishnan, and M. Paterson. Approximating rectangle tiling and packing. 9^{th} *ACM-SIAM Symposium on Discrete Algorithms*, pp. 384-393, 1998.

[6] S. Muthukrishnan, V. Poosala and T. Suel. Rectangular Partitionings: Algorithms, Complexity and Applications. *International Conference on Database Theory (ICDT)*, Springer LNCS, Vol 1540, pp 236–256, 1999.

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

[8] J. Sharp. Tiling Multi-dimensional Arrays. *Foundations of Computing Theory*, Springer, LNCS, Vol 1684, pp. 500-511, 1999.

[9] A. Smith and S. Suri. Rectangular Tiling in Multi-dimensional Arrays. 10^{th} *ACM-SIAM Symp on Discrete Algorithms*, pp 786-794, 1999.