Rectangle Tiling Binary Arrays

Pratik Ghosal
Indian Institute of Technology, Palakkad, India

Syed Mohammad Meesum
Krea University, India

Katarzyna Paluch
University of Wrocław, Wrocław, Poland

Abstract

The problem of rectangle tiling binary arrays is defined as follows. Given an $n \times n$ array A of zeros and ones and a natural number p, our task is to partition A into at most p rectangular tiles, so that the maximal weight of a tile is minimized. A tile is any rectangular subarray of A. The weight of a tile is the sum of elements that fall within it. We present a linear ($O(n^2)$) time $(\frac{3}{2} + \frac{p^2}{w(A)})$-approximation algorithm for this problem, where $w(A)$ denotes the weight of the whole array A. This improves on the previously known approximation with the ratio 2 when $\frac{p^2}{w(A)} < 1/2$.

The result is best possible in the following sense. The algorithm employs the lower bound of $L = \lceil \frac{w(A)}{p} \rceil$, which is the only known and used bound on the optimum in all algorithms for rectangle tiling. We prove that a better approximation factor for the binary RTile cannot be achieved using L, because there exist arrays, whose every partition contains a tile with weight at least $(\frac{3}{2} + \frac{p}{w(A)})L$. We also consider the dual problem of rectangle tiling for binary arrays, where we are given an upper bound on the weight of the tiles, and we have to cover the array A with the minimum number of non-overlapping tiles. Both problems have natural extensions to d-dimensional versions, for which we provide analogous results.

2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases Rectangle Tiling, RTILE, DRTILE

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2024.28

Category APPROX

Related Version *Full Version*: `https://arxiv.org/pdf/2007.14142`

Funding This work was partially funded by Polish National Science Center grant UMO-2018/29/B/ST6/02633.

1 Introduction

In this paper we study several variants of the rectangle tiling problem. These problems belong to a very wide class of discrete optimization tiling problems. As an input, we are given a two-dimensional array $A[1...n, 1...n]$, where each cell $A[i, j]$ has a non-negative weight.

RTile. Given a two-dimensional array A of size $n \times n$ and a natural number p, we partition A into at most p rectangular subarrays, called *tiles*, so that the maximum weight of any tile is minimized. In other words, we have to cover A with tiles such that no two tiles overlap, while minimizing the weight of any tile. The weight of a tile is the sum of the elements that fall within it.

DRTile. A natural variant of RTile is called the DRTile problem. The DRTile problem is a dual of the RTile problem, where we are given an upper bound W on the weight of the tiles, and we have to cover the array A with the minimum number of non-overlapping tiles.

These two problems have a natural extension to d dimensions. Here the input is a d-dimensional array A of size n in each dimension and we have to partition A into non-overlapping d-dimensional tiles so that the optimality criterion of the RTile/DRTile problem is satisfied.

In this paper we consider a special case of the RTile/DRTile problem, where each cell has a binary weight, i.e., the weight of any cell is either 0 or 1. We extend our approach to solve the d-dimensional binary RTile/DRTile problem.

Motivation. The RTile/ DRTile problem is a general problem in combinatorial optimization that has a wide variety of applications in real life. These include load balancing in parallel computing environments, video compression, data partitioning, database mining, and building equisum histogram on two or more attributes. A detailed description of the practical applications of RTile/ DRTile problem can be found in [1, 7, 11].

Related Work. Both the RTile and DRTile problems can be solved in polynomial time when the array is one-dimensional. The RTile problem can be solved using dynamic programming in time $O(np)$. For any fixed $\epsilon < 1$, the best known algorithm has the running time $O(\min\{n + p^{1+\epsilon}, n \log n\})$ [8]. An extensive survey on the RTile problem in one-dimension can be found in [8]. On the other hand, the DRTile problem in one dimension can be solved using a greedy algorithm in linear time.

Both the RTile and DRTile problems have been proven to be NP-hard [7]. Grigni and Manne [6] proved that optimal $p \times p$ tiling (which is a restricted variant of the RTile problem) is NP-hard even when the cell weight is binary. Charikar et al. [3] showed this problem to be APX-hard and NP-hard to approximate within a factor of 2. Khanna et al. [7] proved the RTile problem to be NP-hard to achieve a $\frac{5}{4}$-approximation. Recently Głuch and Loryś [5] have improved the lower bound of the RTile problem to $\frac{4}{3}$. It is not known whether the binary RTile is solvable in polynomial time or NP-hard. Khanna et al. [7] gave the first approximation algorithm for the RTile problem with the ratio $\frac{5}{2}$. The approximation ratio was improved to $\frac{7}{3}$ independently by Sharp [15] and Loryś and Paluch [9]. Loryś and Paluch [10] gave a $\frac{9}{4}$-approximation algorithm for this problem. Berman et al. [1] improved the approximation ratio to $\frac{11}{5}$. Finally, Paluch [13] gave a $\frac{17}{8}$-approximation for this problem and also proved that the approximation ratio is tight with respect to the used lower bound. As far as the DRTile problem is concerned, Khanna et al. [7] gave an $O(n^5)$-time 4-approximation algorithm using the Hierarchical Binary Tiling (HBT) technique. They improved the approximation ratio to 2 using a modified version of the HBT technique, but the running time of this algorithm is very high making the algorithm less practical. Loryś and Paluch [9] also gave a 4-approximation for the DRTile problem while improving the running time to linear.

The d-dimensional version of this problem was introduced by Smith and Suri [16]. They gave a $\frac{d+3}{2}$-approximation algorithm that runs in time $O(n^d + p \log n^d)$. Sharp [15] improved the approximation ratio to $\frac{d^2+2d-1}{2d-1}$ that runs in time $O(n^d + 2^d p \log n^d)$. Paluch [14] gave a $\frac{d+2}{2}$-approximation algorithm while matching the previous running time. She also proved that the ratio is tight with respect to the known lower bound of the problem.

RPack is an extensively studied variant of rectangle tiling, in which we are given a set of axis-parallel weighted rectangles in a $n \times n$ grid, and the goal is to find at most k disjoint rectangles of largest weight. Khanna et al. [7] proved that this problem is NP-hard even when each rectangle intersects at most three other rectangles. They gave an $O(\log n)$-approximation algorithm for RPack that runs in $O(n^2 p \log n)$ time. In [1] Berman et al.

considered the multi-dimensional version of this problem. The dual of RPack is known to be NP-hard even when we are interested in finding a sub-set of disjoint rectangles with a total weight equal to at least some given w. Du et al. [4] considered a min-max version of RTile, where the weight of each tile cannot be smaller than the given lower bound and the aim is to minimize the maximum weight of a tile. They [4] gave a 5-approximation algorithm for this problem and Berman and Raskhodnikova [2] improved the approximation factor to 4 and the approximation ratio of the binary variant to 3.

Previous Work. The binary version of the RTile problem has also been studied. Khanna et al. [7] gave a $\frac{9}{4}$-approximation for the binary RTile problem. Loryś and Paluch [9] and Berman et al. [1] independently improved the approximation ratio for binary RTile to 2.

Our Results. We improve the approximation ratio of the binary RTile problem to $\frac{3}{2} + \frac{p^2}{w(A)}$, where $w(A)$ denotes the number of ones in A. For the arrays A satisfying $\frac{p^2}{w(A)} \approx 0$, it implies that the approximation ratio of the algorithm amounts to $\frac{3}{2}$. The running time of our algorithm is linear ($O(n^2)$). The approximation is best possible in the following sense. The algorithm employs the lower bound of $L = \lceil \frac{w(A)}{p} \rceil$, which is the only known and used bound on the optimum in all algorithms for rectangle tiling. We prove that a better approximation factor for the binary RTile cannot be achieved using L, because there exist arrays, whose every partition contains a tile of weight at least $(\frac{3}{2} + \frac{p}{w(A)})L$.

The general approach to solving this problem is to some extent similar to the approach of [13]. The found tiling is also hierarchical and we use the notions of *boundaries* and *types of columns/subarrays* as well as we apply linear programming in a non-standard way. However, in the present paper the types of subarrays are organized in a somewhat different manner. In particular, the idea of *shadows* is new. To compute the desired partition of A into tiles, we only check a small number of tilings of simply defined subarrays. The subarrays are identified with the help of so called *boundaries* and their *shadows*, which, roughly speaking, designate parts of A tileable in a certain manner and having a weight greater than $\frac{3}{2}L$. To prove the tileability of subarrays composed of multiple simpler subarrays we employ linear programming. Its application here differs from the one in [13] in that each dimension is treated completely symmetrically and thus more "globally" and in the method of showing the feasibility of dual programs. We show that the binary DRTile problem can be approximated by reducing it to the binary RTile problem. As for the d-dimensional binary RTile problem, the algorithm for the 2-dimensional binary RTile problem can be extended to obtain an approximation for the d-dimensional binary RTile problem. The same approximation ratio for the d-dimensional binary DRTile problem can also be found analogously.

Organization. In Section 2, we recall the necessary definitions. In Section 3, we revisit the definition of a boundary and introduce shadows of a boundary. In Section 4, we assume that $w(A) \gg p^2$ and present a $\frac{3}{2}$-approximation algorithm for the RTile problem. The goal of this section is also to introduce the methods needed for the approximation of the binary RTile more gradually, without obscuring the presentation with many technical aspects. In Section 5 we present a $(\frac{3}{2} + \frac{p^2}{w(A)})$-approximation algorithm for the general case (which, in particular, applies also when $\frac{p^2}{w(A)}$ is not negligible). This approximation is achieved by applying only small modifications to the approach described in Section 4. In Section 6, we show that the approximation factor we obtain for the RTile problem is tight with respect to the known lower bound. Section 7 contains our result on the DRTile problem. We conclude by presenting an approximation algorithm for the multi-dimentional RTile problem in Section 8.

2 Preliminaries

Let A be a two-dimensional array of size $n \times n$, where each of its elements belongs to the set $\{0, 1\}$. Given A and a natural number p, we want to partition A into p rectangular subarrays, called *tiles* so that the maximal weight of a tile is minimized. The weight of a tile T, denoted $w(T)$, is the sum of elements within T. $w(A)$ denotes the weight of the whole array A. Since any array element is either equal to 0 or 1, $w(A)$ amounts to the number of 1s in A.

First, notice that the problem is trivial when $p \geq w(A)$. Assume then that $p < w(A)$. Clearly, the maximal weight of a tile cannot be smaller than $\frac{w(A)}{p}$. Consequently, $L = \lceil \frac{w(A)}{p} \rceil$ is a lower bound on the value of the optimal solution to the RTile problem.

Thus to design an α-approximation algorithm for the RTile problem, it suffices to demonstrate the method of partitioning A into p tiles such that the weight of each tile does not surpass αL.

The number p of allowed tiles is linked to the weight of the array A in the following manner.

▶ **Fact 1.** *Let $w(A)$ and L be as defined above. Then, $p \geq \lceil \frac{w(A)}{L} \rceil$.*

The proof directly follows from the assumption that $L = \lceil \frac{w(A)}{p} \rceil$

▶ **Definition 2.** *An array A is said to be f-partitioned if it is partitioned into rectangular tiles such that the weight of any tile does not exceed f.*

We denote by $A[i]$ the i-th column of A, by $A[i..j]$ a subarray of A consisting of columns $i, i+1, \ldots, j$. Thus $A^T[i]$ denotes the i-th row of A and $A^T[i..j]$ a subarray of A consisting of rows $i, i+1, \ldots, j$.

3 The Boundaries and Their Shadows

Let us assume that we want to design an α-approximation algorithm for the RTile problem. Hence the weight of any tile must not exceed αL. In other words, we want to obtain an αL-partitioning for A.

To help find such a partitioning we are going to make use of a sequence of (vertical) *boundaries* and their *shadows*. The vertical boundaries and shadows of array A are defined iteratively below. Each boundary and each shadow is a distinct column of A. The i-th boundary of A is denoted as $B_i = A[b_i]$, i.e., B_i is the b_i-th column of A (or equivalently, $B_i = A[k]$, where $k = b_i$). Similarly, the i-th shadow of A is denoted as $B'_i = A[b'_i]$. The number of boundaries and their shadows depends on the weight and structure of A. The shadow $B'_i = A[b'_i]$ is equal to either B_i or the column succeeding B_i, i.e. either $b'_i = b_i$, or $b'_i = b_i + 1$. For each boundary B_i we define its *type* - we say that boundary B_i is of type j, denoted as $t(B_i) = j$, if its weight satisfies $\lfloor \frac{w(B_i)}{\alpha L} \rfloor = j - 1$.

The ideas behind boundaries and shadows are as follows. The first vertical boundary B_1 indicates simply which part of the array consisting of successive columns starting from the leftmost, exceeds αL. This means that such a subarray cannot be covered with one tile. However, the subarray $A[1..b_1 - 1]$ ending on column $b_1 - 1$ can form one tile, because its weight is not greater than αL. For $i > 1$ the i-th boundary $B_i = A[b_i]$ is established in the following way. We distinguish two cases: (i) $B_{i-1} = B'_{i-1}$ and (ii) $B_{i-1} \neq B'_{i-1}$. Let us first consider case (i). Suppose that $t(B_{i-1}) = j$. Any boundary of type j can be αL-partitioned (horizontally) into j tiles. We check how far to the right we are able to extend one of such partitions. Thus, to identify the ith boundary B_i, we find b_i such that the subarray

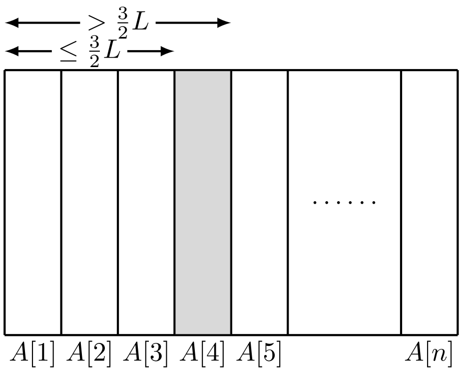


Figure 1 An array with column $A[4]$ as the only boundary.

$A[b'_{i-1}..b_i - 1]$ can be αL-partitioned horizontally into j tiles and the subarray $A[b'_{i-1}..b_i]$ cannot. When (ii) $B_{i-1} \neq B'_{i-1}$, to identify the ith boundary B_i, we proceed in the same way as with the first boundary B_1, i.e., we find b_i such that the subarray $A[b'_{i-1}..b_i - 1]$ can be αL-partitioned horizontally into 1 tile and the subarray $A[b'_{i-1}..b_i]$ cannot.

As for the i-th shadow B'_i we put it in the same column as the boundary B_i if the subarray $A[b'_{i-1}..b_i]$ cannot be αL-partitioned into $t(B_i)$ tiles and otherwise, we put it just behind B_i - in column $b_i + 1$. Notice that in the case of a shadow we check the tileability into $t(B_i)$ tiles and not $t(B_{i-1})$. Also, we observe that $B_i \neq B'_i$ can happen only when $t(B_i) > t(B_{i-1})$ or $B_{i-1} \neq B'_{i-1}$. If $B_i \neq B'_i$, then it means, as we later prove, that the subarray $A[1..b_i]$ is rather easy to partition and we could in fact tile it with a proper number of tiles and start the process of tiling anew with the subarray $A[b'_i..n]$.

We now give a formal definition of a sequence of (vertical) *boundaries* of A and their *shadows*. For technical reasons we introduce a column $A[0]$ to array A.

▶ **Definition 3.** *A boundary B_i is of type j, denoted as $t(B_i) = j$, if its weight satisfies $\left\lfloor \frac{w(B_i)}{\alpha L} \right\rfloor = j - 1$. Based on that, the boundaries and their shadows are defined as follows:*

1. $B[0] = A[0], B'_0 = A[1]$, *thus* $b_0 = 0$ *and* $b'_0 = 1$,
2. ***i*-th boundary B_i :**
   a. *If* $B_{i-1} = B'_{i-1}$, *then* $B_i = A[b_i]$ *iff* $A[b_{i-1}..b_i - 1]$ *can be αL-partitioned horizontally into $t(B_{i-1})$ tiles and* $A[b_{i-1}..b_i]$ *cannot.*
   b. *If* $B_{i-1} \neq B'_{i-1}$, *then* $B_i = A[b_i]$ *iff* $w(A[b_{i-1} + 1..b_i - 1]) \leq \alpha L$ *and* $w(A[b_{i-1} + 1..b_i]) > \alpha L$.
3. ***i*-th Shadow B'_i:** *Let* $t(B_i) = j$.
   $B'_i = B_i$ *iff* $A[b'_{i-1}..b_i]$ *cannot be αL-partitioned horizontally into j tiles.*

The horizontal boundaries are defined analogously. To illustrate the notion of boundaries and shadows let us consider a few examples.

▶ **Example 4.** Array A has only one vertical boundary $B_1 = A[4]$ of type 1.
This means that the total weight of the first 3 columns does not exceed αL, i.e., $w(A[1..3]) \leq \alpha L$, and the weight of the subarray consisting of columns $1 \ldots 4$ does - $w(A[1..4]) > \alpha L$. Since $t(B_1) = 1$, by the definition, the weight of $B_1 = A[4]$ is not greater than αL and the shadow B'_1 of B_1 coincides with B_1. Since A has only one boundary, it means that the weight of the subarray consisting of all columns except for the first 3 is not greater than αL, i.e., $w(A[4..n]) \leq \alpha L$.

▶ **Example 5.** Array A has only one vertical boundary $B_1 = A[4]$ of type 2 and $B'_1 = A[4]$. Exactly as in the example above, we have $w(A[1..3]) \leq \alpha L$ and $w(A[1..4]) > \alpha L$. The weight of B_1 satisfies: $3L \geq w(A[4]) > \alpha L$, because $t(B_1) = 2$. By the fact that $B'_1 = B_1$, we know that the horizontal partition of $A[1..4]$ into 2 tiles of weight not surpassing αL is impossible. Since A has only one boundary, we obtain that $A[4..n]$ can be partitioned into $t(B_1) = 2$ tiles.

▶ **Example 6.** Array A has only one vertical boundary $B_1 = A[4]$ of type 2 and $B'_1 = A[5]$. Again, we have $w(A[1..3]) \leq \alpha L$ and $w(A[1..4]) > \alpha L$. This time, however, $B'_1 \neq B_1$, therefore $A[1..4]$ can be partitioned into 2 horizontal tiles with weight αL at most. Since A has only one boundary and $B'_1 \neq B_1$, we have that $w(A[5..n]) \leq \alpha L$.

▶ **Lemma 7.** *Let k denote the number of vertical boundaries of A and $T_v = \sum_{i=1}^{k} t(B_i)$. Then array A can be αL- tiled with $T_v + 1$ tiles.*

Proof. Suppose first that for each $1 \leq i \leq k$ it holds that $B_i = B'_i$. Then by Definition 3, each subarray $A[b_i..b_{i+1} - 1]$ can be tiled horizontally with $t(B_i)$ tiles and the subarray $A[1..b_1 - 1]$ can be covered by 1 tile. Therefore we indeed use $T_v + 1$ tiles.

For the general case, let $i = \min\{k : B_k \neq B'_k\}$. It means that the subarray $A[b_{i-1}..b_i]$ can be tiled horizontally with $t(B_i)$ tiles. By Definition 3 for each $j \leq i - 2$ the subarray $A[b_j..b_{j+1} - 1]$ can be tiled horizontally with $t(B_j)$ tiles and the subarray $A[1..b_1 - 1]$ can be covered by a single tile. This way the number of used tiles amounts to $\sum_{j=1}^{i-2} t(B_j) + t(B_i) + 1 \leq \sum_{j=1}^{i} t(B_j)$. We continue in the same manner with the subarray $A[b'_i..n]$. ◀

Analogously, we define a horizontal sequence of boundaries of A, i.e., a vertical sequence of boundaries of A^T.

Throughout the paper, $B_1, B_2, \ldots, B_k$ and $C_1, \ldots, C_l$ denote, respectively, the vertical and horizontal sequence of boundaries of A. Let $T_v = \sum_{i=1}^{k} t(B_i)$ and $T_h = \sum_{i=1}^{l} t(C_i)$ and let $T = \min\{T_v, T_h\}$.

▶ **Fact 8.** *Array A can be αL-tiled with $T + 1$ tiles.*

Since we can always αL-partition A into $T + 1$ tiles, to prove that there exists an α-approximation algorithm for the binary RTile problem, it suffices to show that $T + 1$ is an allowed number of tiles, i.e., that $T + 1 \leq p$. To do so, it is enough to prove that it always holds that $w(A) > TL$. This is because since $w(A) \leq pL$ and T and p are integers, the inequality $w(A) > TL$ implies $T + 1 \leq p$.

We state this observation as:

▶ **Fact 9.** *Let α be such that for any A it holds that $w(A) > TL$. Then $p \geq T + 1$ and there exists an α-approximation algorithm for the binary* RTile *problem.*

Let us first note that it is easy to prove that $w(A) > \frac{TL}{2}$.

▶ **Lemma 10.** *The weight of A satisfies $w(A) > \frac{TL}{2}$. Hence, $p > \frac{T}{2}$.*

Proof. We begin by proving that it is always possible to partition the array A vertically into disjoint subarrays $A_1, A_2, \ldots, A_k$ where each subarray except the first (A_1) will contain one of the following boundary types:

1. A single boundary of type j, where j is greater than one.
2. Two boundaries of type 1.
3. A boundary of type j followed by a boundary of type 1, where j is greater than one.

The first subarray A_1 may have any of the types of boundaries mentioned above or a single boundary of type 1. We will now describe how to construct these subarrays.

Let $B_1, B_2, \ldots, B_l$ be a sequence of vertical boundaries of A. We construct the subarrays $A_1, \ldots, A_k$ iteratively. If $t(B_l) > 1$, then we define $A[b_l..n]$ as the last vertical subarray, otherwise, the last subarray is represented by $A[b_{l-1}..n]$. We then repeat this process on the remaining array $A[1...b_{x-1}]$, where $x \in \{l-1, l\}$ based on our choice of the last vertical subarray. We continue until we cannot construct a vertical subarray with one of the sets of boundaries mentioned in points $1-3$.

In this case, the remaining subarray either has no boundary or has a boundary of type 1. If it has no boundary, we merge it with the vertical subarray containing the first boundary. If it has a boundary of type 1, we define it as the first vertical subarray and call it A_1.

Suppose T_i represents the sum of the types of boundaries that are located within the subarray A_i. Our goal is to prove that $w(A_i) \geq \frac{1}{2}T_iL$. If A_i contains a boundary B_r of type $j > 1$ then,

$$w(A_i) \geq w(B_r) \geq (j-1).\frac{3}{2}L \geq \frac{1}{3}(j+1).\frac{3}{2}L$$
$$\geq \frac{1}{2}t(B_r)L = \frac{1}{2}T_iL$$

If A_i contains two boundaries B_r and B_{r+1} of type 1 then,

$$w(A_i) \geq w(A[b_r \ldots b_{r+1}]) \geq \frac{3}{2}L$$
$$\geq \frac{1}{2}(t(B_r) + t(B_{r+1}))\frac{3}{2}L > \frac{1}{2}T_i.L$$

If B_r is a boundary of type $j > 1$ and B_{r+1} is a boundary of type 1 in A_i then,

$$w(A_1) \geq w(B_r) \geq (j-1).\frac{3}{2}L \geq \frac{1}{3}(j+1).\frac{3}{2}L$$
$$\geq \frac{1}{2}(t(B_r) + t(B_{r+1}))L = \frac{1}{2}T_i.L$$

If the A_1 contains one of the above sets of boundaries then $w(A_1) \geq \frac{T_1.L}{2}$. Otherwise, A_1 contains a single boundary of type 1. Then,

$$w(A_1) \geq \frac{3}{2}L > \frac{1}{2}t(B_1)L = \frac{1}{2}T_1.L$$

In conclusion we have proved that for each A_i, $w(A_i) \geq \frac{T_i.L}{2}$. Therefore $w(A) \geq \frac{TL}{2}$. ◀

We now present a lemma that establishes conditions under which a subarray A' of A can be partitioned into horizontal tiles.

▶ **Lemma 11.** *Let $A' = A[i_1 \ldots i_2]$ be a subarray of A and k a natural number greater or equal 2.*

Suppose that $\frac{w(A[i_1]) + w(A[i_2])}{k} + w(A[i_1+1 \ldots i_2-1]) \leq \alpha L$. Then A' can be partitioned into k horizontal tiles of weight at most: (i) $\alpha L + 1$ if $k = 2$, (ii) $\alpha L + 2$ if $k > 2$.

Proof. We divide the number equal to the sum of the weights of two columns $A[i_1]$ and $A[i_2]$ (the columns may be unconnected) by k. We check where the division lines fall with respect to the subarray. Often they may occur in the middle of a row (consisting of two array elements) and we have to move the division so that the whole row is included or the

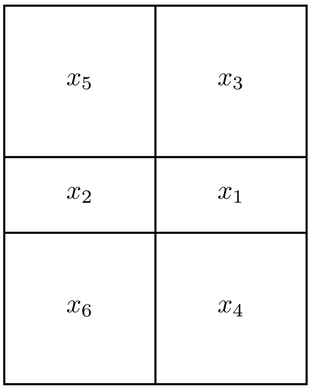


Figure 2 An array with one horizontal boundary containing the subarrays x_2 and x_1 and one vertical boundary containing the subarrays x_3, x_1 and x_4.

whole row is excluded. If $k = 2$, then we choose one of the two options - moving the division upwards or downwards, hence, in the worst case we may have to increase the weight of one tile by 1. For $k > 2$, we may have to shift the division by almost the whole row and thus increase the weight of some tiles by 2. Next we extend this partition to include the subarray $A'' = A[i_1 + 1 \dots i_2 - 1]$ - we do not change the partition of the two-column subarray, but simply follow the partition lines. In the worst case the whole weight of A'' will fall into only one tile - yielding a tile of weight $\frac{w(A[i_1])+w(A[i_2])}{k} + w(A[i_1 + 1 \dots i_2 - 1])$. ◀

4 A $\frac{3}{2}$-approximation when $w(A) \gg p^2$

In this section we deal with arrays such that $\frac{p^2}{w(A)}$ is close to 0, which means that the total weight of any p^2 elements of A is negligible. We are going to show that under this assumption, for $\alpha = \frac{3}{2}$, the weight of A satisfies $w(A) > TL$. Hence, by Fact 9 we get that for this type of arrays there exists a $\frac{3}{2}$-approximation for the binary RTile problem. For the general case the proof that $\alpha = \frac{3}{2} + \frac{p^2}{w(A)}$, the weight of A satisfies $w(A) > TL$, presented in the next section will be only a slight modification of the one shown here.

▶ **Convention 12.** *Throughout this section whenever we speak about* **tiling** *and* **partitioning**, *we respectively mean "$\frac{3}{2}L$-partitioning" and "tiling using tiles of weight at most $\frac{3}{2}L$".*

▶ Remark. The total number of cells at the intersection of the horizontal and vertical boundaries is $O(T^2)$. By Lemma 10 we have that $O(T^2) = O(p^2)$. Therefore by the assumption of this section, it implies that the total weight of the cells in the intersections of the boundaries is negligible with respect to the total weight of the array.

▶ **Observation 13** ([12]). *Assume we have two complexes: one as in Figure 2 and the other with the variables related to the variables of the first one as follows: $x'_1 = x_1$, $x'_4 = x_4$, $x'_3 = x_3$, $x'_5 = x'_6 = 0$ and $x'_2 = x_2 + \max\{x_5, x_6\}$. Then the weight of the second complex is not bigger than the weight of the first one while the inequalities describing the first complex remain true for the second.*

Given an array A we build a linear program, with the help of which we will be able to relate the total weight of the array to the sum of types of boundaries T, i.e., we will show that $w(A) > TL$.

Using Observation 13, we can assume that the whole weight of the array A is contained in the boundaries, i.e., each element of A that does not belong to any boundary has value 0. Each vertical boundary B_i is crossed by l horizontal boundaries and thus cut into $l+1$ parts. We assign a variable $x_{j,i}$ to each part, i.e., the jth part of B_i consists of elements $A[c_{j-1}+1, b_i], A[c_{j-1}+2, b_i], \ldots, A[c_j - 1, b_i]$ and $x_{j,i}$ denotes the sum of the weights of these elements. Similarly, each horizontal boundary C_i is crossed by k vertical boundaries and thus cut into $k+1$ parts. We assign a variable $z_{i,j}$ to each such part. The value of each variable $x_{j,i}$ or $z_{i,j}$ denotes the weight of the corresponding part of the boundary.

In the linear program, we minimize the sum of non-negative variables $x_{j,i}$ and $z_{i,j}$ subject to a set of constraints associated with the boundaries. For each vertical boundary B_i we will have either one or two constraints of the following form:

1. If $t(B_i) > 1$, then we add the constraint $\frac{1}{t(B_i)-1}\sum_{j=1}^{l+1} x_{j,i} \geq \frac{3}{2}L$, which simply describes the total weight of B_i.
2. **a.** $B'_{i-1} \neq B_{i-1}$.
   - **i.** $t(B_i) = 1$. The added constraint is $\sum_{j=1}^{l} z_{j,i} + \sum_{j=1}^{l+1} x_{j,i} > \frac{3}{2}L$.
   - **ii.** $t(B_i) > 1$ and $B'_i = B_i$ (which means that $A[b'_{i-1}..b_i]$ cannot be tiled horizontally with $t(B_i)$ tiles). By Lemma 11 we are justified to add the constraint $\sum_{j=1}^{l} z_{j,i} + \frac{1}{t(B_i)}\sum_{j=1}^{l+1} x_{j,i} > \frac{3}{2}L$.
   - **iii.** $t(B_i) > 1$ and $B'_i \neq B_i$. In this case we do not add any constraint.

   b. $B'_{i-1} = B_{i-1}$.
   - **i.** $B'_i \neq B_i$. The added constraint is $\sum_{j=1}^{l} z_{j,i} + \frac{1}{t(B_{i-1})}(\sum_{j=1}^{l+1} x_{j,i} + \sum_{j=1}^{l+1} x_{j,i-1}) > \frac{3}{2}L$.
   - **ii.** $B'_i = B_i$. Let $T_i = \max\{t(B_{i-1}), t(B_i)\}$. The constraint we add is $\sum_{j=1}^{l} z_{j,i} + \frac{1}{T_i}(\sum_{j=1}^{l+1} x_{j,i} + \sum_{j=1}^{l+1} x_{j,i-1}) > \frac{3}{2}L$. The constraint is a consequence of Lemma 11.

Thus, each B_i defines either one or two constraints. Analogously, each horizontal variable C_j also defines one or two constraints. The linear program dual to the one we have just described has dual variables y'_i, y_i. For each B_i with $t(B_i) > 1$ let y'_i denote the dual variable corresponding to the constraint $\frac{1}{t(B_i)-1}\sum_{j=1}^{l+1} x_{j,i} > \frac{3}{2}L$. The other type of a constraint (if it exists) defined by B_i is represented by y_i. The dual variables corresponding to horizontal boundaries are w_i, w'_i.

▶ **Example 14.** In this example array A has one vertical boundary B_1 of type 1 and one horizontal boundary C_1 of type 1.

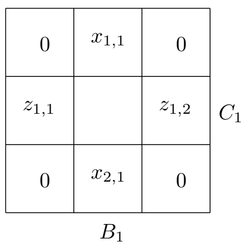


Figure 3 An array with one vertical and one horizontal boundary.

The linear program for A looks as follows. In brackets we give the dual variables corresponding to respective inequalities.

$$\begin{aligned} \text{minimize} \quad & x_{1,1} + x_{2,1} + z_{1,1} + z_{1,2} \\ \text{subject to} \quad & x_{1,1} + x_{2,1} + z_{1,1} > \tfrac{3}{2}L \quad (y_1) \\ & x_{1,1} + z_{1,1} + z_{1,2} > \tfrac{3}{2}L \quad (w_1) \end{aligned}$$

0	$x_{1,1}$	0	$x_{1,2}$	0	
$z_{1,1}$		$z_{1,2}$		$z_{1,3}$	C_1
0	$x_{2,1}$	0	$x_{2,2}$	0	
$z_{2,1}$		$z_{2,2}$		$z_{2,3}$	C_2
0	$x_{3,1}$	0	$x_{3,2}$	0	
	B_1		B_2		

Figure 4 An array with two vertical and two horizontal boundaries.

▶ **Example 15.** In this example array A has two vertical boundaries and two horizontal ones, each of the four boundaries is of type 1. The array is depicted in Figure 4. The linear program for A looks as follows. In brackets we give the dual variables corresponding to respective inequalities.

$$
\begin{array}{ll}
\text{minimize} & \sum_{i=1}^{2}\sum_{j=1}^{3} x_{j,i} + \sum_{i=1}^{2}\sum_{j=1}^{3} z_{i,j} \\
\text{subject to} & \sum_{j=1}^{3} x_{j,1} + \sum_{i=1}^{2} z_{i,1} > \frac{3}{2}L \quad (y_1) \\
& \sum_{i=1}^{2}\sum_{j=1}^{3} x_{j,i} + \sum_{i=1}^{2} z_{i,2} > \frac{3}{2}L \quad (y_2) \\
& \sum_{j=1}^{3} z_{1,j} + \sum_{i=1}^{2} x_{1,i} > \frac{3}{2}L \quad (w_1) \\
& \sum_{i=1}^{2}\sum_{j=1}^{3} z_{i,j} + \sum_{i=1}^{2} x_{2,i} > \frac{3}{2}L \quad (w_2)
\end{array}
$$

▶ **Example 16.** In this example array A has two vertical boundaries B_1, B_2 and two horizontal ones C_1, C_2. Their types are the following: $t(B_1) = t(C_2) = 2$ and $t(B_2) = t(C_1) = 1$. Also $B_1' \neq B_1$ and $C_2' = C_2$.

$$
\begin{array}{ll}
\text{minimize} & \sum_{i=1}^{2}\sum_{j=1}^{3} x_{j,i} + \sum_{i=1}^{2}\sum_{j=1}^{3} z_{i,j} \\
\text{subject to} & \\
& \frac{1}{2}\sum_{j=1}^{3} x_{j,1} + \sum_{i=1}^{2} z_{i,1} > \frac{3}{2}L \quad (y_1) \\
& \sum_{j=1}^{3} x_{j,1} > \frac{3}{2}L \quad (y_1') \\
& \sum_{i=1}^{2} z_{i,2} + \sum_{j=1}^{3} x_{j,2} > \frac{3}{2}L \quad (y_2) \\
& \sum_{j=1}^{3} z_{1,j} + \sum_{i=1}^{2} x_{1,i} > \frac{3}{2}L \quad (w_1) \\
& \frac{1}{2}\sum_{i=1}^{2}\sum_{j=1}^{3} z_{i,j} + \sum_{i=1}^{2} x_{2,i} > \frac{3}{2}L \quad (w_2) \\
& \sum_{j=1}^{3} z_{2,j} > \frac{3}{2}L \quad (w_2')
\end{array}
$$

Let us now build dual linear program for the primal linear program of the Example 16. The dual linear program has the form:

$$
\begin{array}{ll}
\text{maximize} & \frac{3}{2}L(y_1 + y_2 + w_1 + w_2) \\
\text{subject to} & y_1' + \frac{1}{2}y_1 + w_1 \leq 1 \quad (x_{1,1}) \\
& y_2 + w_1 \leq 1 \quad (x_{1,2}) \\
& y_1' + \frac{1}{2}y_1 + w_2 \leq 1 \quad (x_{2,1}) \\
& y_2 + w_2 \leq 1 \quad (x_{2,2}) \\
& y_1 + w_1 + \frac{1}{2}w_2 \leq 1 \quad (z_{1,1}) \\
& y_2 + w_1 + \frac{1}{2}w_2 \leq 1 \quad (z_{1,2}) \\
& y_1 + w_2' + \frac{1}{2}w_2 \leq 1 \quad (z_{2,1}) \\
& y_2 + w_2' + \frac{1}{2}w_2 \leq 1 \quad (z_{2,2})
\end{array}
$$

Algorithm 1 for the binary RTILE problem.

1: $A \leftarrow [1 \dots n, 1 \dots n]$ a two-dimensional array
2: Construct the horizontal and vertical boundaries and their shadows using Definition 3.
3: $B \leftarrow \{B_1, B_2, \dots B_k\}$ (the vertical boundaries, where each $B_i = A[b_i]$)
4: $B' \leftarrow \{B'_1, B'_2, \dots B'_k\}$ (the shadows of B_is, where each $B'_i = A[b'_i]$)
5: $t(B) \leftarrow \{t(B_1), t(B_2), \dots t(B_k)\}$ (the types of the vertical boundaries)
6: $C \leftarrow \{C_1, C_2, \dots C_l\}$ (the horizontal boundaries, where each $C_i = A[c_i]$)
7: $C' \leftarrow \{C'_1, C'_2, \dots C'_l\}$ (the horizontal boundaries, where each $C'_i = A[c'_i]$)
8: $t(C) \leftarrow \{t(C_1), t(C_2), \dots t(C_l)\}$ (the types of the horizontal boundaries)
9: $T_v \leftarrow \sum_{i=1}^{k} t(B_i)$
10: $T_h \leftarrow \sum_{i=1}^{l} t(C_i)$
11: **if** $T_v \leq T_h$ **then**
12:     use the vertical boundaries B as described below from line 15
13: **else**
14:     use the horizontal boundaries C instead of the vertical ones
15: **if** $B_k = B'_k$ **then**
16:     partition $A[b_k \dots n]$ horizontally into $t(B_k)$ tiles
17: **else**
18:     cover $A[b_k + 1 \dots n]$ with one tile
19: **for** $i = k - 1 \dots, 1$ **do**
20:     **if** $B_i = B'_i$ **then**
21:         **if** $B_{i+1} = B'_{i+1}$ **then**
22:             tile $A[b_i \dots b_{i+1} - 1]$ horiz. into $t(B_i)$ tiles (by point 2a of Definition 3)
23:         **else**
24:             partition $A[b_i \dots b_{i+1}]$ horiz. into $t(B_{i+1})$ tiles (by point 3 of Definition 3)
25:     **else**($B_{i+1} \neq B'_{i+1}$)
26:         **if** $B_{i+1} = B'_{i+1}$ **then**
27:             cover $A[b_i + 1 \dots b_{i+1} - 1]$ horiz. with one tile (by point 2b of Definition 3)
28:         **else**
29:             partition $A[b_i + 1 \dots b_{i+1}]$ horiz. into $t(B_{i+1})$ tiles (by point 3 of Definition 3)
30: **if** $B_1 = B'_1$ **then**
31:     cover $A[1 \dots b_1 - 1]$ with one tile
32: **else**
33:     partition $A[1 \dots b_1]$ horiz. into $t(B_1)$ tiles

To figure out the form of constraints constituting the dual program in general, let us consider a variable $x_{j,i}$. Notice that it occurs in at most one constraint defined by a horizontal boundary. It can possibly be contained only in the constraint defined by C_j represented by w_j, where its coefficient is 1. If $t(B_i) = 1$, then we do not have y'_i and $x_{j,i}$ occurs in the constraint represented by y_i and possibly in the constraint represented by y_{i+1}. Thus the inequality in the dual program corresponding to $x_{j,i}$ has the form $\alpha_{j,i} y_i + \alpha_{j,i+1} y_{i+1} + \beta_{j,i} w_j \leq 1$, where each of the coefficients belongs to $[0, 1]$.

If $t(B_i) > 1$, then we do have y'_i and $x_{j,i}$ occurs in this constraint with the coefficient $\frac{1}{t(B_i)-1}$. If $B_i \neq B'_i$, then $x_{j,i}$ does not occur in any other constraints and the inequality in the dual program has the form $\frac{1}{t(B_i)-1} y'_i + \beta_{j,i} w_j \leq 1$, where $\beta_{j,i} \in [0, 1]$. Otherwise, $x_{j,i}$ may also belong to the constraints represented by y_i and y_{i+1}. In each one of them it occurs with the coefficient equal to at most $\frac{1}{t(B_i)}$. Therefore the inequality has the form $\frac{1}{t(B_i)-1} y'_i + \alpha_{j,i} y_i + \alpha_{j,i+1} y_{i+1} + \beta_{j,i} w_j \leq 1$, where each of the coefficients $\alpha_{j,i+1}, \alpha_{j,i}$ belongs to $[0, \frac{1}{t(B_i)}]$.

We are ready to lower bound the weight of the array A with the aid of its boundaries and their shadows.

▶ **Lemma 17.** *$w(A) > TL$.*

Proof. Since the value of any cost function of the dual linear program described above is a lower bound on the minimal value of the cost function of the primal linear program, it suffices to find a feasible assignment of the dual variables such that the cost function will be have value greater than TL.

▷ Claim 18. We can satisfy all constraints of the dual program by assigning the following values to the dual variables. Each y_i and each w_j is assigned $\frac{1}{3}$. If B_i defines only one constraint y'_i, then we assign $\frac{t(B_i)}{3}$ to y'_i. Otherwise y'_i is assigned $\frac{t(B_i)-1}{3}$.

The claim follows from the fact that the inequality $\frac{1}{T-1} \cdot \frac{T}{3} + \frac{1}{3} \leq 1$ is satisfied by each $T \geq 2$ and that the inequality $\frac{T-1}{3(T-1)} + \frac{2}{3T} + \frac{1}{3} \leq 1$ is also satisfied by each $T \geq 2$.

This means that the total value contributed by the dual variables y_i, y'_i (corresponding to constraints defined by boundary B_i) is at least $\frac{t(B_i)}{3}$.

Thus $w(A) > \frac{3}{2}L(\frac{T_v}{3} + \frac{T_h}{3}) \geq TL$. ◀

We show a method for finding a tiling of A with at most $T+1$ tiles. By Facts 1, 8, Lemmas 7 and 17, we proved that $T + 1 \leq p$, when the approximation factor is $\frac{3}{2}$. In other words, we obtain a $\frac{3}{2}$-approximation algorithm for binary RTile.

▶ **Theorem 19.** *For any array A satisfying $\frac{p^2}{w(A)} \approx 0$, there exists a linear time $\frac{3}{2}$-approximation algorithm for binary* RTile.

5 A $(\frac{3}{2} + \beta)$-approximation

In this section we examine the general case, arrays such that $\frac{p^2}{w(A)}$ is not negligible. We will aim for a $(\frac{3}{2} + \beta)$-approximation. When $\beta < \frac{1}{2}$, the approximation ratio of our algorithm is better than 2. Throughout the section, whenever we refer to tiling and partitioning, we mean $(\frac{3}{2} + \beta)L$-partitioning and tiling using tiles of weight at most $(\frac{3}{2} + \beta)L$.

We define a sequence of boundaries and shadows analogously as in the previous section, but with respect to $(\frac{3}{2} + \beta)L$, i.e., we replace each occurrence of "$\frac{3}{2}L$" with "$(\frac{3}{2} + \beta)L$" and modify the meaning of tiling and partitioning accordingly, i.e., to $(\frac{3}{2} + \beta)L$-partitioning.

We want to prove an analogue of Lemma 17. To this end we will consider an analogous linear program, in which we have all the variables occurring in the previous section and additionally we have a variable $s_{i,j}$ for each pair (B_i, C_j), which denotes the element of A at the intersection of the vertical boundary B_i and the horizontal boundary C_j. The function we minimize is $\sum_{i=1}^{k} x_{j,i} + \sum_{j=1}^{l} zi, j + \sum_{i=1}^{k} \sum_{j=1}^{l} s_{i,j}$. For each variable $s_{i,j}$ we have an additional constraint: $-s_{i,j} \geq -1$. The variable $s_{i,j}$ is also included in all those constraints which refer to the part of A covering the intersection of B_i with C_j.

For instance, the linear program for the array from Example 14 is modified as follows:

$$\begin{array}{lrl}
\text{minimize} & x_{1,1} + x_{2,1} + z_{1,1} + z_{1,2} + s_{1,1} & \\
\text{subject to} & x_{1,1} + x_{2,1} + z_{1,1} + s_{1,1} > (\frac{3}{2} + \beta)L & (y_1) \\
& x_{1,1} + z_{1,1} + z_{1,2} + s_{1,1} > (\frac{3}{2} + \beta)L & (w_1) \\
& -s_{1,1} \geq -1 & (t_{1,1}).
\end{array}$$

The linear program for the array from Example 15 in the new scenario looks as follows:

$$\begin{array}{lrl}
\text{min} & \sum_{i=1}^{2} \sum_{j=1}^{3} x_{j,i} + \sum_{i=1}^{2} \sum_{j=1}^{3} z_{i,j} + \sum_{i=1}^{2} \sum_{j=1}^{2} s_{i,j} & \\
\text{s.t.} & \sum_{j=1}^{3} x_{j,1} + \sum_{i=1}^{2} z_{i,1} + \sum_{j=1}^{2} s_{1,j} > (\frac{3}{2} + \beta)L & (y_1) \\
& \sum_{i=1}^{2} \sum_{j=1}^{3} x_{j,i} + \sum_{i=1}^{2} z_{i,2} + \sum_{i=1}^{2} \sum_{j=1}^{2} s_{i,j} > (\frac{3}{2} + \beta)L & (y_2) \\
& \sum_{j=1}^{3} z_{1,j} + \sum_{i=1}^{2} x_{1,i} + \sum_{i=1}^{2} s_{i,1} > (\frac{3}{2} + \beta)L & (w_1) \\
& \sum_{i=1}^{2} \sum_{j=1}^{3} z_{i,j} + \sum_{i=1}^{2} x_{2,i} + \sum_{i=1}^{2} \sum_{j=1}^{2} s_{i,j} > (\frac{3}{2} + \beta)L & (w_2) \\
& -s_{i,j} \geq -1 (t_{i,j}), \text{ for each } 1 \leq i, j \leq 2. &
\end{array}$$

Correspondingly, in the dual program we maximize $(\frac{3}{2}+\beta)L(\sum y_i + \sum w_j) - \sum t_{i,j}$ and we have an additional constraint for each primal variable $s_{i,j}$.

The dual linear program for the array from Example 15 contains the following additional inequalities.

$$\begin{aligned}
&y_1 + y_2 + w_1 + w_2 - t_{1,1} \le 1 && (s_{1,1})\\
&y_1 + y_2 + w_2 - t_{1,2} \le 1 && (s_{1,2})\\
&y_2 + w_1 + w_2 - t_{2,1} \le 1 && (s_{2,1})\\
&y_2 + w_2 - t_{2,2} \le 1 && (s_{2,2}).
\end{aligned}$$

We can see that if we want to assign $\frac{1}{3}$ to each variable y_i, w_j, then we sometimes also have to assign $\frac{1}{3}$ to variables $t_{i,j}$ to ensure the feasibility - compare the first inequality in the set of additional inequalities above. We can notice that we have to assign $\frac{1}{3}$ to $t_{i,j}$ only if both $i < k$ and $j < l$, i.e. when $s_{i,j}$ does not belong to B_k or C_l.

▶ **Lemma 20.** *For $\beta = \frac{p^2}{w(A)}$, it holds that $w(A) > TL$.*

Proof. We can satisfy all constraints of the dual program by assigning the following values to the dual variables. Each y_i and each w_j is assigned $\frac{1}{3}$. If B_i defines only one constraint y'_i, then we assign $\frac{t(B_i)}{3}$ to y'_i. Otherwise y'_i is assigned $\frac{t(B_i)-1}{3}$. Also, each $t_{i,j}$ such that $i < k$ and $j < l$ is assigned $\frac{1}{3}$.

Some of the constraints in the primal program have value $(\frac{3}{2}+\beta)L - 2$ on the right hand side. Such constraints correspond to some borders of type greater than 2, when we use Lemma 11. Let us analyze such cases in more detail. Assume that for a boundary B_i of type $k > 2$ we indeed use Lemma 11. Then the primal linear program contains a constraint with value $(\frac{3}{2}+\beta)L - 2$ on the right hand side. This constraint corresponds to the dual variable y_i. We notice that B_i also defines a constraint of type 1, which has $(\frac{3}{2}+\beta)L$ on the right hand side and corresponds to the dual variable y'_i. Hence each such boundary contributes at least $\frac{t(B_i)-1}{3}(\frac{3}{2}+\beta)L + \frac{1}{3}(\frac{3}{2}+\beta)L - 2) \ge (\frac{(\frac{3}{2}+\beta)L - \frac{2}{3}t(B_i)}{3})$ to the cost function of the dual linear program.

Similarly, some of the constraints in the primal program have value $(\frac{3}{2}+\beta)L - 1$ on the right hand side. Such constraints correspond to some borders of type equal to 2, when we use Lemma 11. Each such boundary contributes at least $(\frac{(\frac{3}{2}+\beta)L - \frac{1}{2}t(B_i)}{3})$ to the cost function of the dual linear program.

Thus the value of the cost function of the dual linear program is lower bounded by $((\frac{3}{2}+\beta)L - \frac{2}{3})(\frac{T_v}{3} + \frac{T_h}{3}) - \frac{(T_h-1)(T_v-1)}{3} \ge TL + p\frac{T_v+T_h}{3} - \frac{2}{3}(T_v+T_h) - \frac{(T_h-1)(T_v-1)}{3}$. Since $L \ge \frac{w(A)}{p}$, we get that $w(A) \ge TL + \frac{p(T_v+T_h)}{3} + \frac{T_v+T_h}{9} - \frac{T_vT_h}{3} - \frac{1}{3}$. Because $p \ge T$, we obtain that $\frac{p(T_v+T_h)}{3} + \frac{T_v+T_h}{9} - \frac{T_vT_h}{3} - \frac{1}{3} \ge \frac{T^2}{3} + \frac{T_v+T_h}{9} - \frac{1}{3} > 0$.

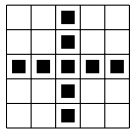
Therefore, $w(A) > TL$. ◀

▶ **Theorem 21.** *There exists a $(\frac{3}{2} + \frac{p^2}{w(A)})$-approximation algorithm for binary* RTile *that has a linear ($O(n^2)$) running time.*

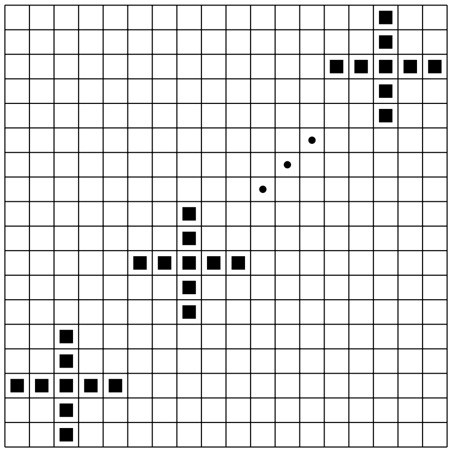
6 Tightness of approximation

In this section, we show that the approximation ratio for the RTile problem is tight with respect to the only known lower bound. Precisely, we prove the following theorem.

▶ **Theorem 22.** *Let $p = 2k$, for some $k \in \mathbb{N}$. Then, there exists a binary array A_k such that the maximum weight of a tile in any tiling of A_k into p tiles has weight at least $\frac{3}{2} \cdot \frac{w(A_k)}{p} + 1$.*



(a)



(b)

Figure 5 The empty squares denote a value of 0, while the ones are colored black. (a) On tiling this array with 3 tiles, one tile will always contain 5 ones, giving an approximation factor of $\frac{5}{3}$. (b) 4-crosses placed in an array for proving an approximation lower bound of $\frac{3}{2}$.

We define an L-*cross* to consist of $2L+1$ ones, it is obtained by taking a $(L+1)\times(L+1)$ array, and filling the $(\frac{L}{2}+1)^{th}$ row as well as the $(\frac{L}{2}+1)^{th}$ column with ones, finally the rest of the entries are filled with zeros. The coordinate $(\frac{L}{2}+1, \frac{L}{2}+1)$ is referred to as the center of the L-cross defined above. An L-cross centered at (x, y) is obtained by translating the center of an L-cross to the coordinate (x, y). Note that an L-cross consists of four contiguous segments of ones, referred to as *arms*, each containing $\frac{L}{2}$ ones. We define A_k as shown in Figure 3.

Proof. Suppose that p is even, therefore $p = 2k$, for some $k \in \mathbb{N}$; our input binary array A_k is obtained as follows. We place k many L-crosses centered at $(j\cdot(L+1)-\frac{L}{2}, j\cdot(L+1)-\frac{L}{2})$, for each $1 \le j \le k$, the rest of the entries of A_k are zero. Note that the L-crosses are placed diagonally in a non-overlapping manner, and $\frac{w(A)}{p} = L$. The array for $L = 4$ is illustrated in Figure 5(a).

If $p = 2$, then it is obvious that one tile will have to contain 3 arms of the cross and thus have weight $\frac{3}{2}L+1$. We will prove that for every $k \in \mathbb{N}$, one of the tiles will have weight at least $\frac{3}{2}L+1$.

Suppose that for k crosses and $2k$ tiles the thesis holds by induction. We will now prove it for $k+1$ crosses and $2k+2$ tiles. Let T_1 be the tile that contains the cell $A_{k+1}[1,1]$. If this tile has weight smaller than $\frac{3}{2}L$, then we have the following two cases:

1. If T_1 does not contain the center of the lower left cross $A_{k+1}[\frac{L}{2}+1, \frac{L}{2}+1]$, then T_1 is formed either by the first $\frac{L}{2}$ columns or the first $\frac{L}{2}$ rows. Due to symmetry, it is enough to consider the case when T_1 is formed by the first $\frac{L}{2}$ columns. Let T_2 be the tile that contains $A_{k+1}[\frac{L}{2}+1, 1]$. If T_2 has weight smaller than $\frac{3}{2}L$, then its upper right corner

$A_{k+1}[x, y]$ is such that either $x < L + 1$ or $y < L + 1$. Due to symmetry, it is enough to consider the case when $x < L + 1$. In this case, if $y < n$, then we can extend T_2 so that $y = n$ without increasing the weight of T_2 as it would not intersect any new L-cross. Thus, we are left with $2k$ tiles, and an array which has A_k as a subarray, therefore by the induction hypothesis we get that the weight of maximum weight tile is at least $\frac{3}{2}L + 1$.

2. If T_1 contains the center of the cross $A_{k+1}[\frac{L}{2} + 1, \frac{L}{2} + 1]$, then its right upper corner $A_{k+1}[x, y]$ is such that $x < L + 1$ or $y < L + 1$. Notice that we may assume that either one tile will be formed by subarray $A_{k+1}[x + 1, 1, n, y]$ or by subarray $A_{k+1}[1, y + 1, x, n]$. This is because the tile that contains $A_{k+1}[x + 1, y + 1]$ cannot contain both $A_{k+1}[1, y + 1]$ and $A_{k+1}[x + 1, 1]$. Suppose w.l.o.g. that the tile T_2 that contains $A_{k+1}[x + 1, 1]$ does not cover any cell of row $y + 1$. We may then extend the upper right corner of T_2 till $A_{k+1}[n, y]$, without increasing the weight of any tile. We are left with $2k$ tiles, and the induction hypothesis gives us the result.

In the case when p is odd, the input binary array A'_p is obtained from A_{k+1} by deleting any rows and columns in it with index at least $k(L + 1) + \frac{L}{2} + 2$. This, in effect adds an extra half L-cross near the upper right corner of A_k. Clearly, for $p = 3$ it is not possible to tile A'_3 with 3 tiles such that the weight of maximum weight tile is less than $\frac{3}{2}L$. The rest of the proof follows from arguments similar to the case when p is even. ◀

▶ Remark. The approximation factor of our algorithm is $\frac{3}{2} + \frac{p^2}{w(A)}$ which is equal to $(\frac{3}{2} + \frac{p}{L})L$. Since $\frac{p}{L}$ is equal to $\frac{p^2}{w(A)}$, it means that the approximation of our algorithm is tight under the condition that $w(A) \gg p^2$.

7 DRTile

In this section, we present an approximation algorithm for the DRTile problem. We have presented a $\frac{3}{2} + \beta$-approximation algorithm for the RTile problem in Section 3. Now we show how to reduce an instance of the DRTile problem to an instance of the RTile problem to achieve an approximation ratio for the DRTile problem. Before we proceed, let us recall the definition of the DRTile problem.

- **the** DRTile **problem**
  - **Input:** A two-dimensional array A and a weight upper bound w.
  - **Goal:** Partition A into a minimum number of non-overlapping tiles, where the weight of each tile must not be larger than W.

Let us consider an array A with $w(A) = n$. Suppose W, provided as input, is the maximum allowed weight of any tile. Clearly, the minimal number of tiles we need to use to cover A is $\lceil \frac{n}{W} \rceil$. Consequently, $\lceil \frac{n}{W} \rceil$ is a lower bound to the optimal solution of the DRTile problem. Our goal is to obtain a γ-approximation algorithm, where γ depends on W. Therefore, the number of tiles we are allowed to use to cover A with this approximation is $\gamma \times \lceil \frac{n}{W} \rceil$.

We construct an instance of the the RTile problem as follows: as an input we have the same array A, and we are allowed to use at most $p = \gamma \times \lceil \frac{n}{W} \rceil$ tiles. Hence from Section 2, the lower bound on the maximum weight of a tile is $\lceil \frac{n}{\gamma \times \lceil \frac{n}{W} \rceil} \rceil$. Hence the maximum weight of a tile with approximation factor of $\frac{3}{2} + \beta$ is,

$$\left\lceil \frac{n}{\gamma \times \lceil \frac{n}{W} \rceil} \right\rceil \times (\frac{3}{2} + \beta) \qquad \leq \left\lceil \frac{n}{\gamma \times \frac{n}{W}} \right\rceil \times (\frac{3}{2} + \beta) \qquad = \left\lceil \frac{W}{\gamma} \right\rceil \times (\frac{3}{2} + \beta).$$

For the solution returned by RTile to be a valid solution of DRTile, the value of $\lceil \frac{W}{\gamma} \rceil \times (\frac{3}{2} + \beta)$ must not exceed W. This allows us to derive a bound on the value of the approximation factor γ, we have,

$$\begin{aligned} &\lceil \frac{W}{\gamma} \rceil \times (\frac{3}{2} + \beta) \le W \\ \Rightarrow &\lceil \frac{W}{\gamma} \rceil \le \frac{W}{(\frac{3}{2} + \beta)} \\ \Rightarrow &\frac{W}{\gamma} \le \frac{W}{(\frac{3}{2} + \beta)} + 1 \\ \Rightarrow &\gamma \ge (\frac{3}{2} + \beta) \cdot \frac{W}{W + (\frac{3}{2} + \beta)}. \end{aligned}$$

This gives us the following theorem.

▶ **Theorem 23.** *There exists a* $(\frac{3}{2} + \beta) \cdot \frac{W}{W+(\frac{3}{2}+\beta)}$*-approximation algorithm for the* DRTile *problem where* $(\frac{3}{2} + \beta)$ *is the approximation factor for the* RTile *problem. The approximation factor of the* DRTile *problem tends to* $\frac{3}{2}$ *as the value of* W *is increased.*

8 The Multidimensional RTile Problem

In Section 3, the algorithm presented for the RTile problem was restricted to two dimensions. In this section, we generalize that algorithm for the d-dimensional RTile problem, where $d \ge 2$. In the d-dimensional RTile problem, we are given a d-dimensional array of size n in each dimension, containing 0/1 as entries, and we have to partition the array into p non-overlapping d-dimensional tiles such that the maximum weight of a tile in a tiling is minimized. Similarly to Section 3, we assume that $\frac{p^d}{w(A)}$ is close to 0 and give a $\frac{2d-1}{d}$-approximation algorithm for the d-dimensional RTile problem. Notice that the approximation ratio converges to 2 as we increase the value of d.

Boundaries and Shadows. The definition of the boundaries and their shadows is a generalization of the definitions in Section 3. The *type* of the boundaries in a d-dimensional array can be defined analogously. By $[i]$, we define the set of boundaries of dimension $n \times n \times \ldots \times 1 \times \ldots \times n$, where i^{th} dimension has size 1.

Let $B_1, B_2, ..B_k \in [i]$, we define $T_i = \sum_{i=1}^{k} t(B_i)$. Finally T is defined as $min\{T_1, T_2, ..., T_d\}$. The following lemma is analogous to Fact 2.

▶ **Lemma 24.** *Let* $T = \{T_1, T_2, ..., T_d\}$*, then the array can be* $\frac{2d-1}{d}$*-tiled with* $T + 1$ *tiles.*

We can estimate the minimal weight of the array using a linear program. The constraints of the linear program have a similar form as mentioned in Section 3. In two dimensional problem, each constraint is greater than $1.5L$. In the d-dimensional RTile problem, each constraint is greater than $\frac{2d-1}{d}L$, instead of $1.5L$.

▶ **Lemma 25.** *Let* $T = \{T_1, T_2, ..., T_d\}$*, then* $w(A) > TL$*.*

The proof of this lemma is analogous to Lemma 20.

▶ **Theorem 26.** *There exists a* $\frac{2d-1}{d}$*-approximation algorithm for the multi-dimensional* RTile *problem assuming* $\frac{p^d}{w(A)}$ *is negligible.*

References

1. Piotr Berman, Bhaskar DasGupta, S. Muthukrishnan, and Suneeta Ramaswami. Improved approximation algorithms for rectangle tiling and packing. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms*, pages 427–436, 2001.
2. Piotr Berman and Sofya Raskhodnikova. Approximation algorithms for min-max generalization problems. *ACM Trans. Algorithms*, 11(1):5:1–5:23, 2014.
3. Moses Charikar, Chandra Chekuri, and Rajeev Motwani. Unpublished. Unpublished manuscript.
4. Wenliang Du, David Eppstein, Michael T. Goodrich, and George S. Lueker. On the approximability of geometric and geographic generalization and the min-max bin covering problem. In *Algorithms and Data Structures, 11th International Symposium, WADS*, pages 242–253, 2009.
5. Grzegorz Gluch and Krzysztof Lorys. 4/3 rectangle tiling lower bound. *CoRR*, abs/1703.01475, 2017. `arXiv:1703.01475`.
6. Michelangelo Grigni and Fredrik Manne. On the complexity of the generalized block distribution. In *Parallel Algorithms for Irregularly Structured Problems, Third International Workshop, IRREGULAR '96*, pages 319–326, 1996.
7. Sanjeev Khanna, S. Muthukrishnan, and Mike Paterson. On approximating rectangle tiling and packing. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, 1998.
8. Sanjeev Khanna, S. Muthukrishnan, and Steven Skiena. Efficient array partitioning. In *Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, 7-11 July 1997, Proceedings*, pages 616–626, 1997.
9. Krzysztof Lorys and Katarzyna E. Paluch. Rectangle tiling. In *Approximation Algorithms for Combinatorial Optimization, Third International Workshop, APPROX2000*, pages 206–213, 2000.
10. Krzysztof Lorys and Katarzyna E. Paluch. New approximation algorithm for RTILE problem. *Theor. Comput. Sci.*, 303(2-3):517–537, 2003.
11. S. Muthukrishnan, Viswanath Poosala, and Torsten Suel. On rectangular partitionings in two dimensions: Algorithms, complexity, and applications. In *Database Theory - ICDT '99*, pages 236–256, 1999.
12. Katarzyna Paluch. *Approximation Algorithms for Rectangle Tiling*. PhD thesis, University of Wrocław, Poland, 2004.
13. Katarzyna E. Paluch. A 2(1/8)-approximation algorithm for rectangle tiling. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004*, pages 1054–1065, 2004.
14. Katarzyna E. Paluch. A new approximation algorithm for multidimensional rectangle tiling. In *Algorithms and Computation, 17th International Symposium, ISAAC 2006*, pages 712–721, 2006.
15. Jonathan P Sharp. Tiling multi-dimensional arrays. In *International Symposium on Fundamentals of Computation Theory*, pages 500–511. Springer, 1999.
16. Adam Smith and Subhash Suri. Rectangular tiling in multidimensional arrays. *J. Algorithms*, 37(2):451–467, 2000.